Analiza metoda umjetne inteligencije u domeni trkaćih igara

Marinić, Hrvoje

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Marinić

ANALIZA METODA UMJETNE INTELIGENCIJE U DOMENI TRKAĆIH IGARA

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Marinić

Matični broj: 0016129838

Studij: Informacijski sustavi

ANALIZA METODA UMJETNE INTELIGENCIJE U DOMENI TRKAĆIH IGARA

ZAVRŠNI RAD

Mentor:

Dr. sc. Bogdan Okreša Đurić

Varaždin, travanj 2023.

Hrvoje Marinić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad se bavi analizom metoda i algoritama strojnog učenja koji su korišteni u dva projekta za izradu trkaće računalne igre. Pružena je teoretska osnova strojnog učenja i pobliže objašnjene pojedinačne metode i algoritmi koji su korišteni. Metoda i algoritam u pitanju su učenje s neuronskim mrežama, koje je podtip nadziranog učenja te NEAT algoritam, koji je podtip neuroevolucije, jednog od poznatijih hibridnih metoda učenja. Također su opisani algoritam gradijenta spusta i njegov podtip algoritam stohastičkog gradijenta spusta. Navedene i opisane su komponente oba projekta, te je provedeno i zabilježeno testiranje metoda i algoritama korištenih u projektima. Zapisana je diskusija o procesu testiranja i dani zaključci. Autor je zaključio kako on smatra NEAT algoritam superiornijim za uporabu u domeni računalnih trkaćih igara, zbog njegove prilagodljivosti, lakoće interpretiranja rezultata i temelja na algoritmu optimizacije.

Ključne riječi: računalne igre, umjetna inteligencija, analiza metoda, strojno učenje, neuronske mreže, NEAT algoritam

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. Strojno učenje 3
   3.1. Definicija i razlozi korištenja strojnog učenja 3
   3.2. Problemi kojima se bavi strojno učenje 4
   3.3. Podjela strojnog učenja 5
4. Nadzirano učenje 8
   4.1. Definicija nadziranog učenja 8
   4.2. Algoritmi i metode nadziranog učenja 10
      4.2.1. Umjetne neuronske mreže 11
5. Učenje s hibridnim algoritmom 24
   5.1. Neuroevolucija 24
      5.1.1. Neuroevolucija augmentirajućih topologija 27
6. Analiza korištenih trkaćih igara 39
   6.1. Analiza projekta Watchcarslearn 39
   6.2. Analiza projekta Micromachine.AI 42
   6.3. Diskusija analiziranih projekata 47
   6.4. Zaključak o analiziranim metodama i algoritmima 48
7. Zaključak 49
Popis literature 50
Reference 50
Popis slika 52

1. Uvod

U ovom radu upoznaje se s osnovama strojnog učenja i analizirati odabrane metode i algoritme umjetne inteligencije koji su primjenjivi u domeni trkaćih računalnih igara. Prvo poglavlje je posvećeno konkretnoj igri u kojoj su testirane odabrane metode i algoritmi umjetne inteligencije; ono sadržava opis igre, tehnike korištene za analizu podataka, popis kontrola koje su bile dostupne agentima, sliku korištene staze i modela auta uz kratke opise njihovih postavljenih karakteristika za vrijeme korištenja.

Za početak razrade teme je tu poglavlje o strojnom učenju, kojemu je svrha pružanje potrebnog predznanja za razumijevanje daljnjih poglavlja u radu. Unutar tog poglavlja pružena je definicija strojnog učenja i problema kojima se ono bavi, kratki popis razloga zbog kojih se koristi strojno učenje, opis njegove primjene u određenim područjima te kako je ono primjenjivo u domeni računalnih igara općenito i specifično u trkaćim računalnim igrama. Također, sadrži podjelu strojnog učenja na dvije osnovne kategorije s kratkim opisom pojedinačnih kategorija i danim primjerima metoda ili algoritama koji pripadaju određenoj kategoriji.

Nakon uvoda u temu, rad nastavlja s analizom pojedinačnih metoda i algoritama umjetne inteligencije i predstavljanjem rezultata koji su proizašli iz testiranja. Za njihovo analiziranje je potrebno razumijevanje njihovih specifičnih definicija koje su pružene na početku uvoda u poglavlje svake metode ili algoritma. Osim pružene definicije, također su navedeni primjeri područja i tipovi poslova za koje se najčešće koristi određena metoda ili algoritam. Objašnjeno je kako se pojedinačna metoda konkretno predstavlja unutar igre koja će biti analizirana, dakle kako umjetna inteligencija odlučuje koje kontrole koristiti, kada i zašto. Objašnjenje slijedi opis odabranih faza i razvoja umjetne inteligencije prema toj specifičnoj metodi ili algoritmu koje su analizirane prema uspješnosti izvršavanja naredbe koja im je zadana. Faze su podijeljene prema vremenu ili broju ciklusa koje je autor proizvoljno odabrao nakon sagledavanja svih sakupljenih informacija. Završetak svakog poglavlja posvećenog pojedinačnoj metodi ili algoritmu se sastoji od donošenja zaključaka prema analizama uspostavljenih faza te uočenih dobrih i loših strana korištenja te metode ili algoritma unutar domene trkaćih računalnih igara.

Ovaj rad završava skupljanjem i analizom zaključaka donesenih u prijašnjim poglavljima kako bi se ustanovilo najefektivniju metodu između dvoje odabranih metoda ili algoritama za korištenje u trkaćim igrama.

2. Metode i tehnike rada

U ovom poglavlju treba opisati koje će metode i tehnike biti korištene pri razradi teme, kako su provedene istraživačke aktivnosti, koji su programski alati ili aplikacije korišteni.

Istraživanje literature je provedeno uz pomoć knjiga, online članaka, odlomaka s konferencija i drugih izvora. Programski alati koji su korišteni za analizu dva projekta su Visual Studio za projekt MicroMachine.AI i Visual Studio Code za pokušaje otvaranja projekta Watchcarslearn. Taj projekt je na kraju testiran na njegovoj web stranici. Iskorištena je stranica desmos.com za kreiranje grafa sigmoidne funkcije, te aplikacija GIMP za uređivanje svih slika kako bi im se dodali hrvatski prijevodi.

3. Strojno učenje

Prije analiziranja pojedinačnih metoda i algoritama strojnog učenja, potrebna je određena razina predznanja iz tog područja. Zato je u ovom poglavlju pružena definicija strojnog učenja, zajedno s problemima koji se pokušavaju riješiti njegovim korištenjem, uz primjere primjene strojnog učenja u nekoliko područja. Podjela strojnog učenja u dvije grube kategorije, podjela u 10 kategorija i kratak opis onih odabranih za analizu služi kao uvod u sljedeća poglavlja.

3.1. Definicija i razlozi korištenja strojnog učenja

Uobičajeno je za rješavanje problema na računalu potreban algoritam. Za isti problem može postojati više primjenjivih algoritama, a na korisniku je da on odabere onaj najprikladniji. I dok to pravilo vrijedi u većini slučajeva, ipak postoje neki problemi za koje nema algoritama te za koje ne bi bilo učinkovito stvarati algoritme. Prepoznavanje govora od strane računala, samovozeći automobili, učenje računala da igraju igre poput šaha ili predviđanje ponašanja kupaca su samo neki od primjera takvih problema. U tim slučajevima je važno da sustav koji se koristi ima sposobnost učenja u promjenjivoj okolini u kojoj se nalazi. Ako sustav može učiti i prilagoditi se promjenama oko njega, onda dizajner sustava ne mora predvidjeti i pružiti rješenje za svaki mogući problem te se za njega može reći da je uistinu prilagodljiv, ako ne inteligentan.

Proces učenja za strojeve je poprilično različit od onog za ljude; strojevi uče oslanjajući se na podatke koji su im dostupni, dok je ljudima prirodnije učiti iz vlastitih prošlih iskustava. Imajući to na umu, strojno učenje se može objasniti na najosnovnijoj razini kao „kategorija umjetne inteligencije koja omogućava računalima da „razmišljaju“ i uče samostalno“ [1, p. 1]. Cilj tog procesa je da „računalo mijenja aktivnosti koje izvodi za ostvarivanje nekog zadatka kako bi poboljšalo njihovu preciznost, gdje bismo preciznost određivali prema mjeri uspješnosti ostvarenja zadatka koje je računalo ostvarilo izvođenjem te aktivnosti“ [1, p. 1].

Strojno učenje je tijekom godina definirano više puta, ali definicija koja se pokazala najkorisnijom za analizu u ovome radu je pružena od strane Tom Mitchella (1997) čiji prijevod glasi: „Za kompjuterski program možemo reći da uči iz iskustva E koje je povezano s nekim zadatkom T i može se mjeriti nekim mjerilom performansi P, ako se njegova performansa na T, mjerljiva sa P, poboljšava iskustvom E“ [2, p. 2].

3.2. Problemi kojima se bavi strojno učenje

Strojno učenje se koristi u problemima koji pripadaju dolje navedenim kategorijama zato što ono omogućava da računalo izvršava i sofisticirane zadatke bez pomoći od strane ljudi, a „temeljeno je na učenju informacija iz zadatka te na temelju njih donositi odluke i predviđati“ [3, p. 591]. Prema N. Bolfu [3], dvije opće kategorije problema kojima se bavi strojno učenje su sljedeće:

- **Zadatci koje uobičajeno izvršavaju ljudi** – Zadatci gdje je izvođenje zadanih naredbi bez greške i praćenje definiranih smjernica glavni problem. Primjeri uključuju kuhanje, vožnju vozila i prepoznavanje govora.
- **Zadatci koje ljudi ne mogu izvršavati** – Zadatci koji uključuju analizu velikog broja kompleksnih podataka gdje je ljudima teško izvući bitne informacije. Primjeri uključuju prognozu vremena, pretraživanje interneta ili web trgovinu.

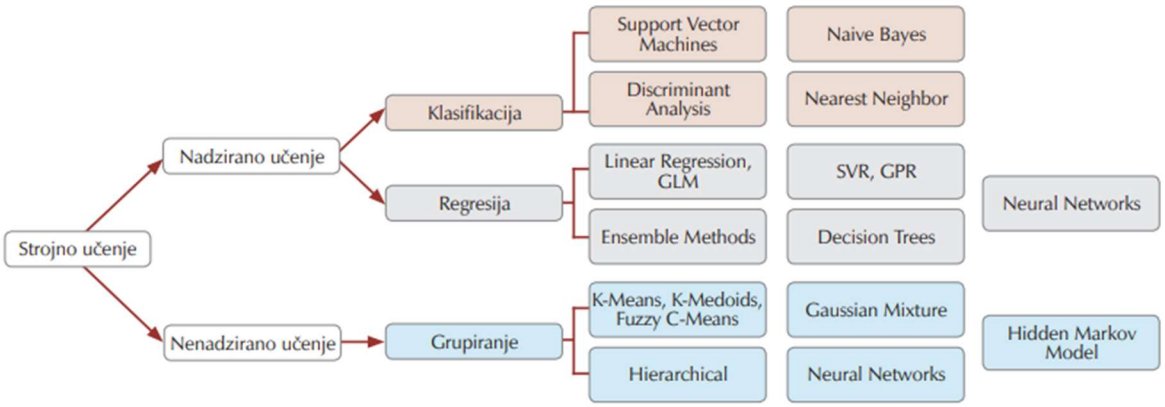
Prije nego dođe do rješavanja problema, on se mora ispravno kategorizirati kako bi se iskoristio odgovarajući algoritam strojnog učenja. Prema Jafaru Alzubi (2018) bilo koji problem se može grupirati u jednu od sljedećih kategorija problema:

- **Problem klasifikacije** – Tip problema gdje je izlaz samo jedan od fiksnog broja predodređenih izlaznih klasa poput Da/Ne, Istina/Laž, ili A/B/C problema. Ovisno o broju izlaznih klasa problem može biti binarni ili multiklasni problem klasifikacije.
- **Otkrivanje nepravilnosti** – Ovoj kategoriji pripadaju oni problemi u kojima se analizira nekakav uzorak u potrazi za promjenama ili nepravilnostima. Banke koriste algoritme za otkrivanje nepravilnosti kako bi pronašle odstupanja od uobičajenog ponašanja nekog od njihovih korisnika i obavijestile ih. Takav tip problema se bavi pronalaskom izuzetaka u uzorku.
- **Problem regresije** – Problemi koji sadržavaju kontinuiran i numerički izlaz klasificiraju se kao problemi regresije, a rješavaju se korištenjem regresijskih algoritama.
- **Problem grupiranja (***engl. Clustering***)** – Grupiranje pripada kategoriji algoritama koji koriste učenje bez ljudskog nadzora. Ti algoritmi pokušavaju uočiti strukture unutar podataka i rasporediti podatke u razrede prema sličnosti njihove strukture. To su problemi gdje se pokušava otkriti struktura iza problema i kako je problem organiziran.
- **Problem poticanog učenja (***engl. Reinforcement learning***)** – Algoritmi poticanog učenja se koriste kada se odluke trebaju temeljiti na prošlim iskustvima stečenim učenjem. Oni pružaju način programiranja agenata u okolini koristeći se konceptima nagrade i kazne bez određivanja kako da se zadatak izvrši. Ovaj tip problema je

učestao u programima koji igraju igre ili programima za kontrolu temperature gdje se postavlja pitanje, „Koji je sljedeći korak za poduzeti“.

3.3. Podjela strojnog učenja

Ovisno o tome kako algoritam uči izvršavati zadatak i dostupnosti izlaza tijekom učenja, pristupi strojnom učenju se mogu podijeliti u 10 kategorija [1]. One uključuju: nadzirano učenje, nenadzirano učenje, poticano učenje, evolucijsko učenje, djelomično nadzirano učenje, učenje ansambl metodom, učenje s umjetnom neuronskom mrežom, učenje dimenzionalnom redukcijom i učenje s hibridnim algoritmom. Ali, prema Nenadu Bolfu [3], strojno učenje se može osnovno podijeliti na nadzirano i nenadzirano učenje, pod koje možemo svrstati ostale kategorije, kao što je prikazano na slici 1. U ovom radu će biti pružen samo kratak opis za većinu navedenih kategorija s većim fokusom na kategorije nadziranog učenja i učenja s hibridnim algoritmom iz koje su algoritmi odabrani za analizu trkaće računalne igre.



Slika 1: Podjela metoda strojnog učenja *[3]*

1. **Nadzirano učenje**

   U ovoj kategoriji, algoritam uči iz primjera ili modula za treniranje koji su pruženi s točnim izlazima. Uz pomoć njih algoritam uči odgovarati preciznije tako da uspoređuje izlaze koje dobije s pruženim izlazima. Za nadzirano učenje se može reći da je ono učenje na primjerima [4].

2. **Nenadzirano učenje**

   Pristup nenadziranog učenja se temelji na prepoznavanju neidentificiranih postojećih uzoraka u podacima kako bi se ustanovila pravila koja vrijede za taj skup podataka.

Učenje se odvija bez eksplicitne povratne informacije od strane korisnika. Ovaj pristup se koristi kada kategorije podataka nisu poznate [4].

3. **Poticano učenje**

   Kod poticanog učenja algoritam dobiva poticaj koji mu poručuje da li je učinio nešto dobro i dobio nagradu, ili nešto loše i dobio kaznu, a sam algoritam mora istraživati i mijenjati svoje postupke kako bi dobio što više nagrada [4].

4. **Evolucijsko učenje**

   Ovaj pristup je dobio inspiraciju od živih organizama koji se prilagođavaju svojoj okolini. Algoritam prati svoje ponašanje i mijenja svoje parametre kako bi za ulaze vraćao odgovarajuće izlaze. Odlučuje koje parametre mijenjati uz pomoć funkcije prikladnosti, koju koristi kako bi se algoritam optimizirao i davao najbolje rješenje za neki problem [5].

5. **Djelomično nadzirano učenje**

   U ovoj kategoriji algoritmi se koriste najboljim stranama nadziranog i nenadziranog učenja. Za te algoritme su ili pružene oznake izlaza (nadzirano učenje) ili nisu (nenadzirano učenje), dok se ovi algoritmi koriste u situacijama kad su pružene oznake izlaza za dio promatranja, ali za većinu nisu. Korisno je u problemima poput klasifikacije, regresije ili predviđanja [1].

6. **Učenje ansamblom metoda**

   Kategorija u kojoj je model sastavljen od više slojeva koji su naučeni da rješavaju neki uobičajen problem. Nasuprot drugim metodama koje uče tako da konstruiraju samo jednu hipotezu, ansambl metoda uči tako da ih konstruira nekoliko i kombinira ih u model predviđanja. Može se podijeliti na sekvencijalne i paralelne ansambl pristupe.

7. **Učenje s umjetnom neuronskom mrežom**

   Umjetna neuronska mreža je inspirirana biološkim neuronskim mrežama i sastoji se od tri djela: ulaznog sloja, jednog ili više skrivenih slojeva i izlaznog sloja. Umjetne neuronske mreže su sastavljene od međusobno povezanih jedinica kojima se mreža koristi kako bi za neki uzorak ulaza vraćala odgovarajuće izlaze [5]. Ovisno o ponašanju tijekom učenja, mogu se svrstati u: nadzirane neuronske mreže, nenadzirane neuronske mreže i poticane (*engl. reinforcement*) neuronske mreže [1].

8. **Učenje temeljeno na instancama**

   Za razliku od drugih, ova metoda neće jasno definirati ciljanu funkciju iz početnih podataka. Naziva se lijenim učenikom (*engl. lazy learner*) zato što pohranjuje početnu instancu za trening i odgađa generalizaciju dok ne dođe do klasifikacije nove instance [1]. Lijeni učenik određuje ciljanu funkciju drugačije i lokalno za svaku novu klasificiranu

instancu, umjesto globalno za sve instance i zato ga je lakše trenirati, ali mu treba duže da napravi predviđanje [2].

9. **Učenje dimenzionalnom redukcijom**

   Algoritam dimenzionalne redukcije služi u reduciranju broja dimenzija podataka kako bi trošak kalkuliranja bio što manji i izvođenje operacija s jako kompleksnim i velikim količinama podataka bilo moguće. Algoritam smanjuje broj dimenzija podataka reduciranjem suvišnih i nepotrebnih podataka i čišćenjem podataka kako bi omogućio veći stupanj preciznosti rezultata. „Dimenzionalna redukcija se odvija nenadzirano kako bi pretražila i iskoristila implicitnu strukturu u podacima“ [1].

10. **Učenje s hibridnim algoritmom**

    Iako se isprva učenje ansamblom metoda činilo kao rješenje za učestale probleme kompleksnosti kalkulacija, znanstvenici su našli probleme u tom pristupu. Kompliciran ansambli od više klasifikatora otežavaju implementaciju i analizu rezultata [1]. Novi pristupi rješavanju tog problema uključuju hibridizaciju, to jest kombiniranje više vrsta metoda kako bi se stvorio novi sofisticiraniji algoritam [5].

Od svih gore navedenih vrsta metoda, nadzirano učenje je daleko najpopularnije u znanosti [1], pa je ono i jedno od dvije metode odabrane za analizu u ovome radu. Druga metoda je učenje s hibridnim algoritmom, specifično NEAT algoritmom, te su obje pobliže objašnjene prije analize podataka.

4. Nadzirano učenje

Nadzirano učenje je jedan od najpopularnijih pristupa strojnog učenja čije se metode tradicionalno koriste za rješavanje zadataka koji se bave problemima regresije, klasifikacije ili problemima sa strukturiranim izlazima [4]. Taj pristup je karakteriziran time što su skupovi podataka koje promatra nasumični parovi ulaza i izlaza, prema kojima agent uči prepoznavati, a u najboljem slučaju i predviđati, odgovarajuće izlaze za nove ulaze. U ovom poglavlju je pružena osnovna definicija nadziranog učenja uz detaljnije objašnjenje metode koja je korištena u prvom od korištenih projekata, Watchcarslearn, autora Manas Sarpatwar [5]. Također su pruženi i konkretni primjeri primjene metode nadziranog učenja u domeni igara i izvan nje.

4.1. Definicija nadziranog učenja

Tijekom nadziranog učenja, agent proučava skup parova ulaza i izlaza koji su mu pruženi kao testni podaci i na temelju kojih on aproksimira funkciju koja će za neki novi, nepoznati ulaz vratiti odgovarajući izlaz [6]. Prema toj definiciji se može reći da agent nadziranog učenja prvo mora proučiti skup parova ***x*** i ***y*** za koje zna da si međusobno odgovaraju, te prema kojima onda može predviđati odgovarajuće ***y*** varijable za nove i neviđene ***x*** varijable. Štoviše, termin nadzirano učenje proizlazi iz slučaja u kojima je vrijednost ***y*** varijable vrlo teška za odrediti automatski od strane agenta te mu je potrebna asistencija stvarne osobe koja onda pokazuje agentu što treba učiniti kako bi u budućnosti mogao predvidjeti odgovarajuće ***y*** varijable za druge, nepoznate, ***x*** varijable.

Kako bi se olakšalo razumijevanje principa na kojemu nadzirano učenje funkcionira, pruženo je nekoliko primjera korištenja nadziranog učenja u rješavanju relativno jednostavnih problema. Kao početni primjer je iskorišten slučaj u kojemu agent ima zadatak razvrstavanja voća i vraćanja ispravnog naziva voća u pitanju, to jest, agent mora prepoznati voće prema dostupnim osobinama. Ovaj primjer je pronađen u knjizi Artificial Intelligence and Games autora Georgiosa N. Yannakakisa i Juliana Togeliusa [7]. U ovom slučaju agent, kada vidi neko voće, mora moći zaključiti da li je ono kruška ili jabuka ovisno o boji i veličini voća u pitanju te prema toj informaciji vratiti odgovarajuću oznaku kruške ili jabuke. U početku se agent trenira promatranjem skupa parova koji se sastoje od slika kruški i jabuka te njihovih odgovarajućih naziva. Dakle, slika voća je ulazna ili ***x*** varijabla, dok su boja i oblik voća osobine koje agent koristi kako bi vratio odgovarajući naziv, to jest izlaznu ili ***y*** varijablu. Na kraju učenja, u najboljem slučaju, agent mora moći prepoznati koji naziv između jabuke i kruške vratiti za neki novi, dosad neviđeni primjerak jabuke ili kruške koristeći se samo informacijama o njezinoj boji

i obliku. Za drugi primjer je odabran slučaj tangencijalno povezan s trkaćim igrama, gdje je agentov zadatak zaprimiti trenutno stanje automobila, koje se sastoji od njegove trenutne brzine i smjera kretanja te stanja ceste na kojoj se vozi, nakon čega treba vratiti udaljenost koja je bila potrebna za zaustavljanje automobila. Ulazne varijable, ili ***x*** varijable, se sastoje od trenutnog stanja, a izlazne, ili ***y*** varijable, od udaljenosti potrebne da se automobil zaustavi, dok su osobine prema kojima agent zaključuje udaljenost zaustavljanja brzina i smjer automobila te stanje ceste na kojoj se vozi. Kako bi mogao započeti učenje, agentu se pruža skup parova ***x*** i ***y*** varijabli koji sadržava parove poput (50 km/h, auto okrenut prema lijevo, suha cesta; 49 metara) i (75 km/h, auto okrenut ravno, mokra cesta; 123 metara). Nakon proučavanja skupa parova, agent konstruira funkciju kojom može procijeniti da li neki ***y*** izlaz odgovara nekome ***x*** ulazu. Nakon učenja na testnim podacima, agent u najboljem slučaju mora moći odrediti udaljenost zaustavljanja na dotad neviđenom stanju ceste, imajući dostupno samo znanje o brzini i smjeru automobila te stanju ceste na kojoj se vozi. Također je bitno napomenuti da, iako u ovom primjeru agent može sam odrediti točnu ***y*** varijablu nakon što se auto zaustavio, primjer još uvijek spada pod nadzirano učenje.

Imajući na umu gore navedene primjere, moguće je zaključiti da nadzirano učenje zahtijeva početni skup „podatka za treniranje koji se sastoji od ***N*** parova ulaza ***x*** i izlaza ***y***

$$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N),$$

a gdje je svaki od parova ***x*** i ***y*** dobiven korištenjem neke nepoznate funkcije $y = f(x)$, dok agent pokušava napraviti aproksimaciju stvarne funkcije ***f*** koja se zove funkcija ***h***“ [6, p. 671] i koja se koristi kako bi agent mogao predviđati odgovarajuće izlaze ***y*** za nepoznate ulaze ***x***. Dakle, koristeći se informacijama koje su izložene dosad, može se reći da nadzirano učenje počinje od prepoznavanja veze između parova u skupu testnih podataka, nastavlja kreiranjem funkcije koja može imitirati, barem približno, ako ne i potpuno taj odnos između parova ulaza i izlaza, te konačno dolazi do korištenja spomenute funkcije kako bi se određivali odgovarajući izlazi za unesene ulaze uz pomoć svojstva generalizacije.

Određivanje funkcije ***h***, koja se također naziva i hipotezom, se provodi tako da se željena funkcija izvuče iz prostora mogućih funkcija, to jest prostora hipoteza, što može biti sve od skupa polinoma trećeg stupnja, pa do skupa Javascript funkcija [6]. Početna funkcija ***h*** koju agent pronađe ne mora biti, a najčešće i nije, identična funkciji ***f*** te ju je potrebno testirati koristeći se još jednim skupom podataka, prikladno nazvanim testnim skupom, koristeći se mjerilom poput točnosti predviđanja, kako bi se utvrdilo da li funkcija može provoditi generalizaciju na zadovoljavajućoj razini te kako bi se izbjeglo pretjerano treniranje agenta. Pojam pretjeranog treniranja agenta je pojašnjen na sljedećoj stranici. Za funkciju ***h*** se može reći da je zadovoljavajuća, ako uspješno predvidi izlaze za testni skup ili ostvari zadovoljavajući

stupanj točnosti [7]. To naravno vodi do pitanja kako pronaći dobru hipotezu unutar prostora hipoteza. Predloženo rješenje je da se sustav zadovolji sa hipotezom koja nije nužno identična funkciji ***f***, ali uspijeva vratiti izlaz za svaki uneseni ulaz u skupu podataka za treniranje. Zato je bitno naglasiti da poanta nije u traženju savršene funkcije, već one čiji rezultati su najbliži točnim rezultatima za tražene ulaze [6].

Probleme, na koje je moguće naići tijekom traženja zadovoljavajuće hipoteze, a koji su pojašnjeni u ovom paragrafu, se sastoje od pretjeranog treniranja ili prilagođavanja podacima (*engl. overfitting*) te suprotnog problema, premalog treniranja ili prilagođavanja podacima (*engl. overfitting*). Pretjerano treniranje se odnosi na problem u kojemu funkcija u pitanju obraća previše pozornosti na skup podataka koji je korišten za njeno treniranje te tako postane nesposobna za predviđanje odgovarajućih izlaza za dotad neviđene ulaze ili dolazi do broja grešaka tijekom predviđanja koji prelazi granice tolerancije. Suprotni problem, to jest problem nedovoljnog treniranja zna čak nastati kao posljedica izbjegavanja problema pretjeranog treniranja, a on nastaje u slučajevima kada nije bilo dovoljno testnih podataka, testni podaci koji su bili dostupni nisu bili dovoljno kompleksni, ili agentu jednostavno nije pruženo dovoljno vremena da prouči testne podatke. Kao rezultat tog problema, agent postaje nesposoban uočiti ponavljajuće obrazce u testnim podacima uz pomoć kojih bi provodio generalizaciju nad novim i neviđenim ulazima te vraćao odgovarajuće izlaze [6].

4.2. Algoritmi i metode nadziranog učenja

Ponavljajući zaključak iz prethodnog poglavlja, za sve metode i algoritme nadziranog učenja je moguće reći kako dijele istu osnovnu ideju. Ugrubo izražena, ta ideja bi zvučala ovako: „Nadzirani algoritmi učenja su oni koji uče asocirati neki ulaz sa nekim odgovarajućim izlazom, nakon što im je pružen skup parova ulaza ***x*** i izlaza ***y*** na kojemu se treniraju.“ [6]. Usprkos tome što metode i algoritmi nadziranog učenja dijele tu osnovnu ideju, razlike među metodama i algoritmima nadziranog učenja se pojavljuju tijekom izvođenja hipoteze ***h*** iz prostora hipoteza u načinima na koje dolaze do te hipoteze i kako predstavljaju izvedenu funkciju [7]. To naravno vodi do postojanja tolikog broja različitih algoritama nadziranog učenja da je često teško odlučiti koji specifičan algoritam upotrijebiti za neki problem. Djelomično objašnjenje količine različitih algoritama i metoda nadziranog učenja je pruženo u takozvanome „No free lunch“ ili NFL teoremu, koji tvrdi sljedeće: „Bilo koja dva algoritma za optimizaciju su ekvivalentna kada se usporedi prosjek njihove uspješnosti u rješavanju svih mogućih problema“ [8]. Dakle, barem jedan od razloga je taj što trenutno ne postoji takav algoritam nadziranog učenja koji je optimiziran da bude najbolji odabir za bilo koji mogući problem.

U ostatku ovog poglavlja je fokus na jednoj od najpopularnijih metoda nadziranog učenja za razvijanje umjetne inteligencije unutar igara, pogotovo kada agent ima zadatke poput igranja igre, imitacije ponašanja igrača ili predviđanja sklonosti igrača. Metoda učenja o kojoj je riječ je učenje korištenjem umjetne neuronske mreže. Odabrane su umjetne neuronske mreže zbog toga što su one korištene u projektu Micromachine.AI autora Cédrica Bovara [9], koji je jedan od dva projekta analizirana u ovome radu. Također je pobliže pojašnjen algoritam stohastičkog gradijenta spusta koji se koristi za učenje neuronske mreže u projektu, počevši sa kratkim objašnjenjem linearne regresije i gradijenta spusta prije nastavljanja na pojašnjavanje stohastičkog gradijenta spusta.

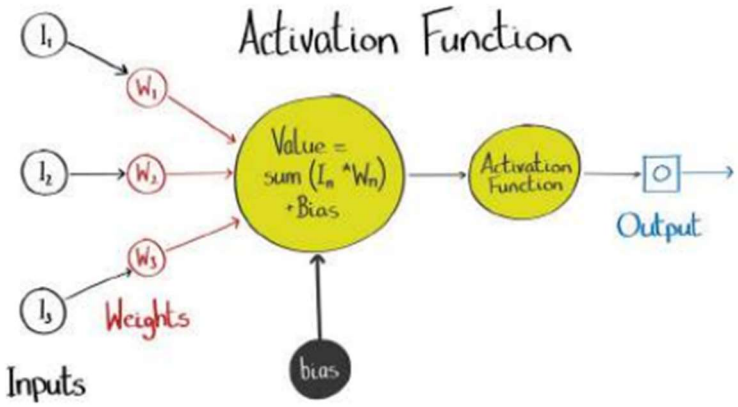
4.2.1.Umjetne neuronske mreže

Kada se govori o umjetnim neuronskim mrežama, također znanim po njihovoj engleskoj kratici ANN (*engl. Artificial Neural Network*), može ih se usporediti sa radom bioloških mozgova od kojih je došla inspiracija za njihovo stvaranje, jer ANN-ovi pokušavaju imitirati način na koji mozgovi obrađuju informacije i funkcioniraju. Tu izjavu podupiru Togelius i Yannakakis u njihovoj knizi Artificial Intelligence and Games [7], gdje govore kako je ANN skup međusobno povezanih neurona, napravljen na način koji oponaša prirodan rad biološkog mozga i tako na sličan način njemu funkcionira i procesira informacije. S druge strane, na ANN-ove je moguće gledati kao na sustave nelinearnih jednadžbi, koji primjenom geometrijskih transformacija na ulaznim podacima pokušavaju dobiti odgovarajuće izlaze [10]. U ANN-ovima se spomenuti neuroni odnose na čvorove od kojih se sastoji ANN, mreža međusobno povezanih čvorova koji služe u pretvaranju ulaznih podataka u izlazne koristeći se primjenom geometrijskih transformacija nad njima. Svaki umjetni neuron ima neki broj ulaza **x**, svaki od kojih je asociran sa svojim parametrom težine **w**. „Parametar težine **w** nekog neurona služi dvije svrhe, povećavanju ili smanjivanju vjerojatnosti okidanja tog neurona“ [11, p. 264]. Prije pretvaranja u izlaz ***y***, potrebno je pomnožiti svaki ulaz **x** sa njegovom odgovarajućom težinom **w**, uzeti sumu tih brojeva i dodati joj težinu pristranosti tog neurona, označenu sa slovom *b*, kao što je prikazano u sljedećoj formuli: $Sum(\boldsymbol{x}_i * \boldsymbol{w}_i) + b$. Tu vrijednost je onda potrebno proslijediti aktivacijskoj funkciji koja će onda vratiti izlaz neurona [11]. Kada se govori o iznad spomenutoj težini pristranosti, pristranost ima otprilike sljedeće značenje: „Sklonost hipoteze koja se koristi za predviđanje da odstupa od očekivanih rezultata kada je sagledana u prosjeku različitih skupova za treniranje“ [6, p. 672]. Imajući te informacije na umu, moguće je donijeti pretpostavku da su umjetne neuronske mreže samo mreža međusobno povezanih matematičkih jednadžba, koje nakon učenja vraćaju hipotezu koja je barem djelomično slična, ako ne i potpuno ista stvarnoj funkciji $y = f(x)$.

ANN-ovi se koriste u brojnim područjima gdje postoji potreba za algoritmima nadziranog učenja, ali „ključna područja primjene ANN-ova uključuju prepoznavanje uzoraka, upravljanje robotima i agentima, igranje igara, donošenje odluka, prepoznavanje gestikulacija, govora i teksta, medicinske i financijske primjene, afektivno modeliranje i prepoznavanje slika“ [7, p. 59].

4.2.1.1. Aktivacijske funkcije

U prethodnom paragrafu je spomenuto kako svaki neuron, nakon što izračuna vrijednost svojih ulaza, mora proslijediti tu vrijednost nekoj aktivacijskoj funkciji koja će onda tu vrijednost transformirati i vratiti izlaznu vrijednost. Prema tome je moguće zaključiti kako aktivacijske funkcije zapravo odlučuju o kakvome je tipu ANN-a riječ, jer se pomoću njih dobivaju odgovarajuće izlazne vrijednosti za unesene ulazne vrijednosti. Stoga je potrebno prvo pojasniti što je to točno aktivacijska funkcija te navesti one koje se najčešće koriste. Najjednostavniju definiciju je pružio Paul Roberts u svojoj knjizi Artificial Intelligence in Games gdje govori kako je „aktivacijska funkcija kada se uzme suma ulaza i provede ju se kroz proces koji će odrediti što će biti izlaz neurona“ [11, p. 264].

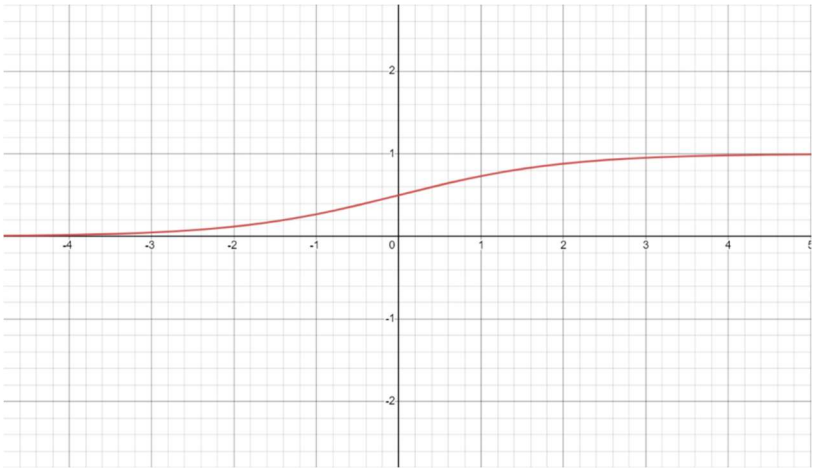


Slika 2:Prošireni prikaz neurona sa aktivacijskom funkcijom*[11]*

Dakle, kao što je vidljivo na slici broj 2, nakon što se svaki od ulaza pomnoži sa svojom odgovarajućom težinom te se njihovom zbroju doda težina pristranosti tog neurona, potrebno je dobivenu vrijednost procesirati koristeći se aktivacijskom funkcijom kako bi se dobio izlaz neurona. U današnje vrijeme postoji više različitih aktivacijskih funkcija koje se koriste zajedno s ANN-ovima i tijekom njihovog treniranja, ali tijekom ovog poglavlja pružena su pojašnjenja prvenstveno za algoritam povratnog rasprostiranja pogreški (*engl. backpropagation*), algoritam stohastičkog gradijenta spusta i logističku aktivacijsku funkciju sigmoidnog oblika.

Logistička funkcija sigmoidnog oblika

Kao uvod u aktivacijske funkcije je prvo opisana ona funkcija koja se najčešće koristi za treniranje ANN-a, a to je „logistička funkcija sigmoidnog oblika $(g(x) = 1/(1 + e^{-x}))$ zbog tri njezine osobine: 1) funkcija je ograničena, jednolična i nelinearna; 2) funkcija je kontinuirana i matematički glatka; 3) funkciji je trivijalno odrediti derivaciju kao $g'(x) = g(x)\big(1 - g(x)\big)$“ [7, p. 60]. Sigmoidnu funkciju je jednostavno porepoznati po karakterističnom „S“ obliku njezinoga grafa, a također je bitno napomenuti da se domena sigmoidne funkcije sastoji od svih realnih brojeva, te da će bez obzira na ulaz, sigmoidna funkcija uvijek vratiti izlaz između vrijednosti 0 i 1. „Numerički je dovoljno računati vrijednosti sigmoidne funkcije koristeći se malim rasponom brojeva, poput brojeva od (-10, 10), zato što ako je broj koji ulazi u funkciju manji od -10, izlaz će imati vrijednost jako blizu 0, a u slučaju da je broj koji ulazi u funkciju veći od 10, onda će izlaz imati vrijednost koja je jako blizu 1“ [12]. Imajući na umu gore navedene osobine, postaje očito zašto je logistička sigmoidna funkcija najčešće korištena aktivacijska funkcija. Dobar dio njenih osobina se može prepoznati na grafičkom prikazu funkcije, kao onome pruženom na slici broj 3. Kao što je vidljivo iz dolje prikazanog, sigmoidna funkcija je ograničena s donje i gornje strane, te joj je raspon rezultata uvijek od skoro 0 do skoro 1, neovisno o vrijednosti realnog ulaznog broja.

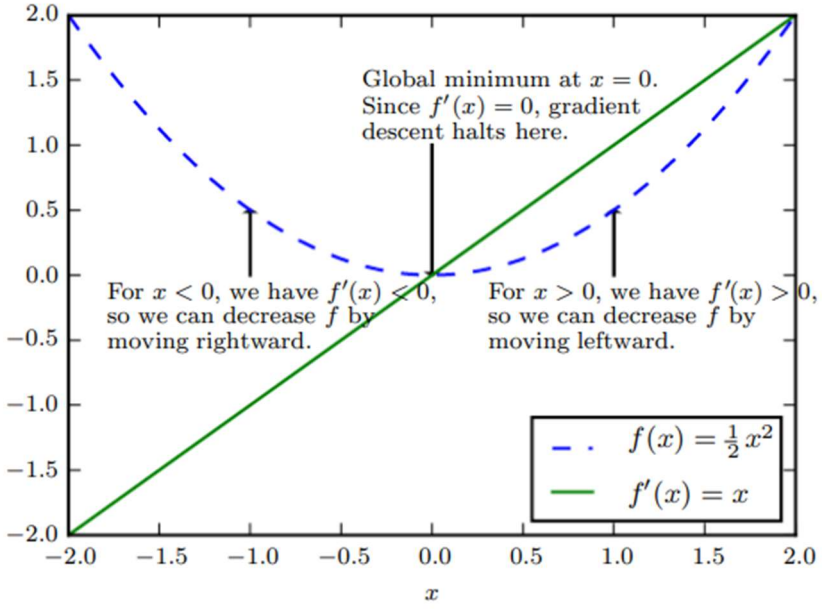


Slika 3: Prikaz logističke sigmoidne funkcije

Algoritam stohastičkog gradijenta spusta

Kako bi se uspješno objasnio algoritam stohastičkog gradijenta spusta prvo je pojašnjen pojam algoritma gradijenta spusta. Kada se govori o algoritmima optimizacije, pod optimizacijom se misli na „zadatak minimiziranja ili maksimiziranja neke funkcije $f(x)$ promjenom varijable ***x***. Problemi optimizacije se uobičajeno gledaju kao problemi minimiziranja $f(x)$, a čak je moguće i prikazati probleme maksimizacije koristeći se algoritmom

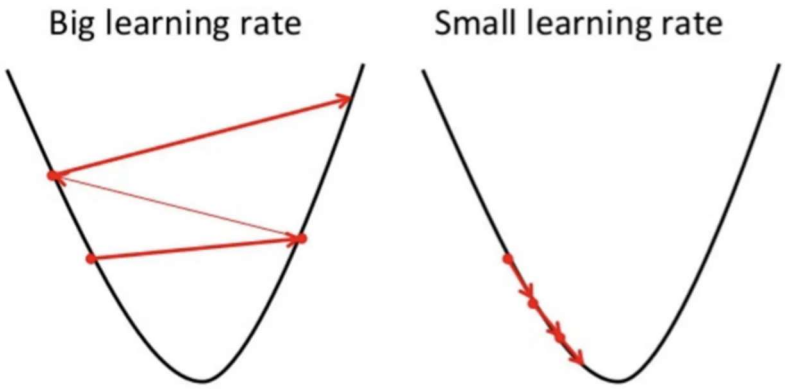
minimiziranja, tako da se minimizira $-f(x)$“ [4, p. 82]. Dakle, moguće je reći kako je algoritam gradijenta spusta, ne samo algoritam optimizacije, nego i algoritam minimiziranja, koji dolazi do rješenja postepenim smanjivanjem varijable ***x*** u funkciji $f(x)$.



Slika 4: Prikaz kako algoritam gradijenta spusta koristi derivacije funkcije za praćenje funkcije do minimuma *[4]*

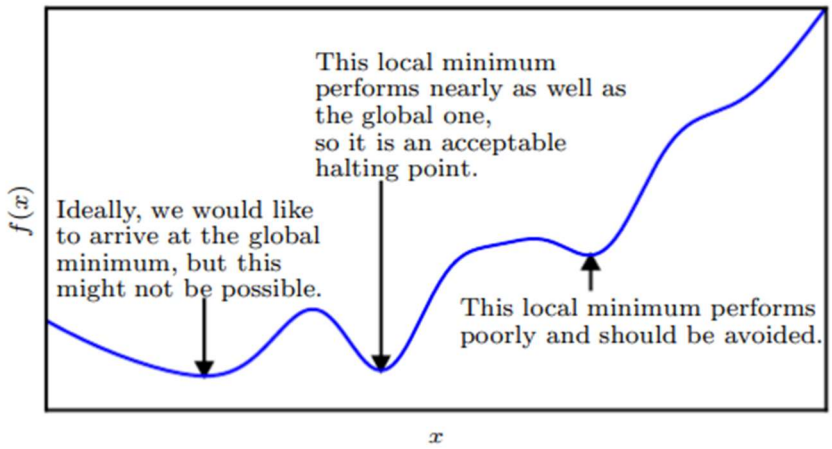
Kao što je vidljivo na slici 4, algoritmu gradijenta spusta je potrebna derivacija funkcije $f(x)$, koja je na slici prikazana punom zelenom linijom, zato što „ona određuje kako se mala promjena u ulazu odražava na odgovarajuću promjenu u izlazu funkcije, što je prikazano sljedećom formulom: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$“ [4, p. 83]. Dakle, moguće je reći kako se derivacija neke funkcije koristi kako bi se saznalo na koji način male promjene ulazne varijable ***x*** djeluju na izlaznu varijablu ***y*** te funkcije. Gradijent spusta je dobio svoje ime po tome što se pokušava smanjiti rezultat funkcije ***f*** „kretanjem“, to jest postepenim umanjivanjem varijable ***x***, od početne točke u smjeru koji ima najstrmiji nagib prema dolje, dakle može se reći spuštanjem, skroz dok se ne dostigne točka gdje su svi elementi gradijenta nula, ili u praksi jako blizu nuli [6]. Ali kako bi se to moglo dogoditi, kao prvi korak je potrebno otkriti nagib funkcije ***f*** u početnoj točki kako bi se znalo u kojemu se smjeru funkcija smanjuje najbrže, da bi se u tome smjeru onda kretalo. Također, treba napomenuti kako je „potrebno izračunati nagib funkcije ***f*** za svaki od pojedinih parametara funkcije ***f*** “ [13]. Drugim riječima, potrebno je izračunati gradijent funkcije ***f***, nakon čega slijedi drugi korak, koji se sastoji od odabira nasumične početne vrijednosti za ulazne varijable ***x*** koje se prosljeđuje u treći korak, svrstavanje odabranih vrijednosti u funkciju ***f***. U četvrtom se koraku izračunava veličina koraka učenja množenjem gradijenta sa stopom učenja. Kao peti korak je potrebno izračunati nove parametre nakon zadnjeg koraka, tako da se stari parametri umanje za veličinu koraka učenja.

Ponavljaju se koraci broja 3-5 skroz dok gradijent ne poprimi vrijednost približnu nuli [13]. Također se treba odabrati stopa učenja, koja će odrediti za koliko će se funkcija pomicati prema minimumu, to jest prema točki gdje će svi dijelovi gradijenta imati vrijednost približnu nuli, tijekom svakog koraka. Problem kod odabira stope učenja leži u tome što, u slučaju da je nastali korak premalen, proces kretanja će trajati predugo i potrošiti previše resursa, dok u slučaju da je nastali korak prevelik, neće se moći doći do minimuma, kao što je prikazano na slici broj 5 [14].



Slika 5: Prikaz razlike u stopama učenja *[14]*

Imajući to sve na umu, moguće je algoritam gradijenta spusta ukratko opisati kao algoritam koji pokušava pronaći parametar ***x*** za koju će neka funkcija ***f*** vratiti minimalni rezultat ***y*** tako što prvo uzima nasumičnu ulaznu vrijednost od koje pokušava doći do minimuma, krećući se u smjeru najstrmijeg nagiba za izračunati korak učenja, dok ne dođe do točke u kojoj će funkcija vraćati vrijednost približnu nuli, kada će onda vratiti korišteni parametar ***x*** za koji funkcija ***f*** vraća izlaznu vrijednost ***y*** koja je približna nuli. Kao primjer bi mogao poslužiti agent s funkcijom koja pokušava predvidjeti cijenu nekog pića, prema cijenama drugih pića čije cijene su mu dostupne kao početni podaci. Dakle, poanta je smanjiti funkciju gubitka preciznosti tako da se ujednače predviđeni izlaz i stvarni izlaz što je više moguće, ili u ovom slučaju, da predviđena cijena bude što je više moguće bliža stvarnoj cijeni. Glavni problemi koji sprečavaju konvergenciju kod gradijenta spusta, su problematičan odabir koraka učenja, problem koji je već pojašnjen, i problem gdje se funkcija ne uspije spustiti do globalnog minimuma, već umjesto toga zapne u jednom od lokalnih minimuma [14]. Pod konvergencijom se misli na proces dolaska do globalnog minimuma funkcije ***f***, to jest dolaska do točke u kojoj algoritam prestaje mijenjati parametre jer je dosegnuta minimalna moguća vrijednost funkcije ***f*** [13]. Zbog problema nemogućnosti spusta do globalnog minimuma, prikazanog na slici broj 6 zajedno sa pojašnjenjem, je najčešće prihvatljivo zaustaviti spuštanje na lokalnom minimumu čija je vrijednost približno jednaka nuli i tako, zaobići problem te uštediti na resursima.



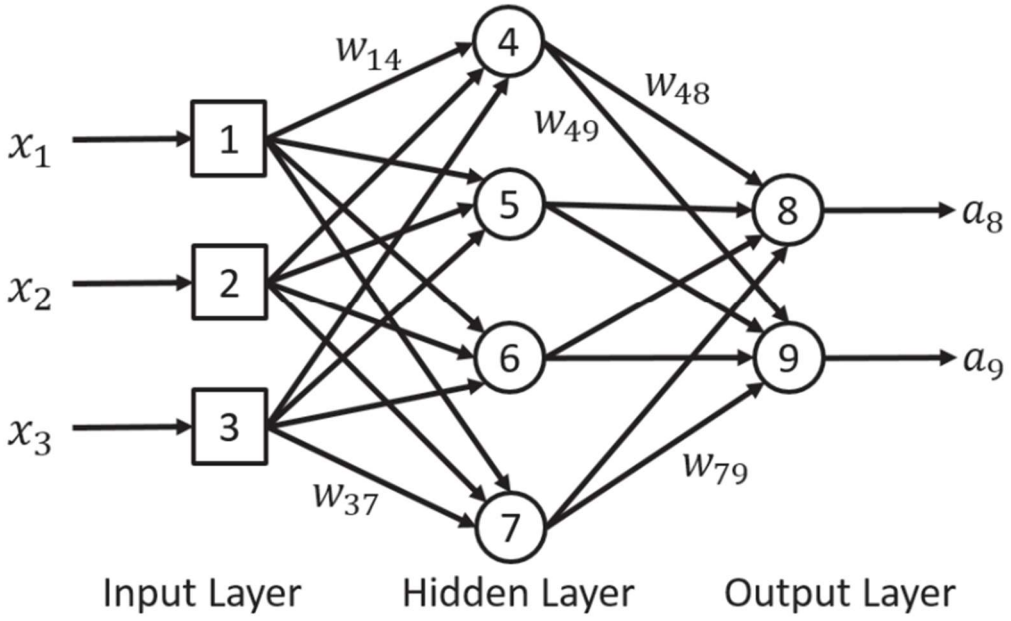
Slika 6: Prikaz problema zapinjanja u lokalnim minimumima *[4]*

Algoritam gradijenta spusta mora izračunati derivaciju funkcije za svaki ulaz ***x*** iz skupa primjera za treniranje, u svakom koraku prema minimumu, što rezultira u ogromnoj količini računanja koja znatno usporava algoritam [6]. Srećom postoji verzija gradijenta spusta zvana stohastički gradijent spusta, gdje se „nasumično odabire mali uzorak ulaznih podataka zvan mini-serija (*engl. mini-batch*) što ima veliki utjecaj na količinu potrebnih proračuna“ [13]. Dakle, stohastički gradijent spusta je samo podvrsta gradijenta spusta, koja uspijeva riješiti proračune brže od osnovnog oblika, što ga čini idealnim za rješavanje problema u kojima postoji veliki broj primjera u skupu podataka za treniranje. „Nije nužno da će uvijek doći ni do lokalnog minimuma u nekom razumnom okviru vremena, ali često uspije pronaći veoma nisku vrijednost funkcije troškova dovoljno brzo da bude koristan“ [4]. Iako nije najpouzdaniji algoritam, isplati ga se koristiti na zadacima s velikim skupovima podataka, a možda ga izbjegavati za jednostavnije zadatke.

4.2.1.2. Sastavljanje neuronske mreže od neurona

Za stvaranje ANN-a, potrebno je organizirati i spojiti neurone ANN-a u funkcionalnu mrežu. Kako bi se to dogodilo, mora se odrediti njegova arhitektura. Pod arhitekturom se misli na „određivanje koliko točno neurona se želi imati u mreži te kako će ti neuroni biti međusobno povezani“ [4, p. 197]. Najjednostavniji oblik ANN-a naziva se višeslojni perceptron, ali za olakšano čitanje će se koristiti njegova engleska kratica MLP (*engl. Multi Layer Perceptron*), unutar MLP-a neuroni su posloženi u grupama zvanim slojevi, gdje je izlaz svakog neurona u nekom sloju spojen sa ulazima svih neurona u sljedećem sloju, ali neuroni u istom sloju nisu međusobno povezani [10]. Dakle, izlazne vrijednosti koje se prosljeđuju neuronima u sljedećem sloju će postati njihove ulazne vrijednosti. Prema tome je moguće logički zaključiti kako su vrijednosti koje proizađu iz izlaza neurona u zadnjem sloju ANN-a također vrijednosti izlaza cjelokupnog ANN-a. Zadnji sloj ANN-a se također naziva izlaznim slojem, za razliku od

slojeva između izlaza i ulaza koji se nazivaju skrivenim slojevima te prvog sloja ANN-a što se zove i ulaznim slojem, kao što je prikazano na slici broj 7. Također je potrebno naglasiti kako „ulazni sloj ne sadržava neurone jer on samo raspodjeljuje ulaze do prvog sloja neurona“ [7, p. 61]. Kada se govori o strukturama MLP-a, glavno je odrediti dubinu mreže, to jest koliko skrivenih slojeva će novi ANN imati, i širinu svakog pojedinačnog sloja, to jest koliko će neurona imati svaki od slojeva [4].



Slika 7: Primjer MLP-a sa tri ulaza, jednim skrivenim slojem koji sadržava četvero skrivenih neurona i dva izlaza *[7]*

Koristeći se gore navedenim informacijama, moguće je ukratko nabrojati osobine MLP-ova:

1) MLP-ovi su slojeviti jer su neuroni koji se nalaze u mreži grupirani u slojeve: ulazni sloj, broj skrivenih slojeva te izlazni sloj
2) MLP-ovi spadaju pod unaprijedne mreže (eng. *feed-forward*) zbog toga što neuroni nisu spojeni međusobno u slojevima, već su spojeni izlazi svakog neurona u nekom sloju s ulazima svakog neurona u sljedećem sloju, zbog čega bi se moglo reći da su uvijek usmjereni prema naprijed (od prethodnog sloja prema sljedećem sloju) [10]
3) „MLP-ovi su potpuno spojeni jer je izlaz svakog neurona u nekom sloju povezan sa ulazom svakog neurona u sljedećem sloju“ [7, p. 61]

Kako bi se olakšalo razumijevanje rada ANN-a, može ga se zamisliti kao sustav kanala za navodnjavanje kroz koji protječe voda koja teče od ulaznog sloja, kanalima kroz neurone, i

tako skroz do izlaznog sloja. Svaki od kanala ima vodena vrata koja predstavljaju parametar težine tog pojedinog kanala. U tom slučaju, ako se neki neuron aktivira puno više od drugih neurona, vodena vrata kanala koji vode prema njemu će biti otvorena do više razine od vrata onih kanala koji vode do neurona koji se ne aktiviraju toliko često. Bitno je napomenuti kako bi voda mogla teći samo u smjeru prema naprijed pa bi se moglo zamisliti da je sustav navodnjavanja na padini čiji nagib sprečava vodu da teče nazad nakon što prođe iz jednog sloja u drugi.

4.2.1.3. Rad prema naprijed

U ovom poglavlju je pojašnjeno kako se izračunava izlaz ANN-a kada je prisutan uzorak ulaza, ali prije toga je pružena osnovna definicija rada prema naprijed (*engl. feed forward*). Paul Roberts je definirao rad prema naprijed kao „proces prolaza podataka kroz mrežu s početkom od ulaznog sloja, i prolaska kroz skrivene slojeve skroz dok se ne dostigne izlazni sloj“ [11, p. 275] u svojoj knjizi Artificial Intelligence in Games. Dakle, proces rada prema naprijed se sastoji od prolaza ulaznih podataka ANN-a kroz njegove uzastopne slojeve kako bi se dobili izlazni podaci.

Prije nego što se može koristiti, potrebno je pripremiti ANN koji radi prema naprijed za početak rada. To se čini označavanjem neurona brojevima i njihovim smještajem u željeni poredak. „Uobičajeno je da se krene brojati od ulaznog sloja i da se brojevi povećavaju do izlaznog sloja“ [7, p. 61]. Također je potrebno napomenuti kako, unatoč tome što se spominju neuroni u ulaznome sloju, te se oni isto broje za svrhe numeriranja, oni zapravo služe samo tome da zaprimljene ulazne vrijednosti ***x*** prenesu dalje do ulaza neurona u sljedećem skrivenom sloju [7]. Primjer takve numeracije se može pronaći na slici broj 6. Zadnji korak prije nego što je moguće koristiti ANN se sastoji od označavanja težina veza između sada raspoređenih i organiziranih neurona. Koristi se w_{ij} kako bi se predstavila težina veze između neurona *i* koji se nalazi u prethodnom sloju te neurona *j* koji se nalazi u sljedećem sloju. Isto tako je potrebno označiti težine pristranosti koje se spajaju na sve neurone *j* osim onih u ulaznome sloju, kao b_j [7]. Unatoč tome, ponekad nije moguće prikazati težine pristranosti svakog neurona, kao što je slučaj prikazan na slici broj 6, jer to može dovesti do situacije u kojoj cijeli graf ANN-a postane nepregledan.

Nakon obavljene pripreme, moguće je koristiti ANN koji radi prema naprijed tako da se započne sa zadavanjem ulaznih vrijednosti ***x*** neuronima koji se nalaze u ulaznome sloju [10]. Zato što se podaci šire prema naprijed u ANN-u koji radi prema naprijed, potrebno je za svaki neuron *j* izračunati njegov izlaz kako bi se prolazom kroz mrežu mogao izračunati krajnji izlaz ANN-a ***y***. Računanje izlaza neurona je već pojašnjeno na slici broj 2, ali će ovdje biti ponovljeno u obliku sljedeće formule: „$\alpha_j = g\left(\sum_i \{w_{ij}\alpha_i\} + b_j\right)$, gdje su α_j izlaz neurona , a α_i ulaz neurona

j te je *g* aktivacijska funkcija korištena za dobitak izlaza [7, p. 62]“. Uobičajeno je da je aktivacijska funkcija zapravo logistička funkcija sigmoidnog oblika, ali u slučaju projekta MicroMachine.AI, autora Cédrica Bovara, [9] ona je zamijenjena stohastičkim gradijentom spusta. Dakle, kako bi se dobio izlaz nekog neurona, potrebno je zbrojiti sumu umnoška svih ulaza neurona s njihovim odgovarajućim težinama i težinu pristranosti tog neurona, te novonastali zbroj proslijediti kao ulaz ***x*** nekoj aktivacijskoj funkciji, koja će vratiti izlaz ***y*** neurona. Kao zadnji korak rada sa ANN-om koji radi prema naprijed ostaje samo napomenuti da su izlazi ***y*** neurona koji se nalaze u izlaznom sloju ujedno i izlazi cjelokupnog ANN-a.

4.2.1.4. Kako umjetna neuronska mreža uči?

Autor Paul Roberts u svojoj knjizi Playing Smart [11], iznosi tvrdnju kako u osnovi postoje dva načina na koje je moguće naučiti ANN nešto. Prvi se koristi evolucijskim algoritmima gdje nakon kreiranja strukture ANN-a i pružanja početnih podataka, ANN uči „postepenim mijenjanjem parametara težine između neurona, skroz dok se ne dobije željeni izlaz“ [11, pp. 65-66], dok kao drugi način navodi korištenje algoritma povratnog rasprostiranja pogreški. U ovome poglavlju je objašnjen pristup algoritma povratnog rasprostiranja pogreški, uz funkciju troškova koja se koristi kao mjerilo njegove kvalitete uspjeha. Osim toga je pruženo objašnjenje i potreba za funkcijom troškova u treniranju ANN-a. „Kako bi ANN mogao provesti aproksimaciju tražene funkcije ***f***, prvo je potrebno pronaći algoritam za treniranje koji će moći prilagoditi težine veza među neuronima ANN-a ***w*** i težine njihovih pojedinačnih pristranosti ***b*** na način da njegovi izlazi odgovaraju željenim izlazima skupa podataka **y** tako da se za neki pruženi ulaz ***x*** dobiva njegov odgovarajući izlaz ***y*** “ [7]. Kako bi se moglo pratiti kvalitetu nekog skupa težina potrebno je imati funkciju troškova koja će služiti kao jasno mjerilo njihove kvalitete.

Funkcija troškova

Funkcija troškova je također zvana i funkcijom pogreške, zato što ona mjeri odstupanja između dobivenog izlaza ANN-a i traženog izlaza ANN-a [11]. Služi kako bi se mogle mjeriti performanse nekog MLP-a, jer bez mjerila nije moguće provoditi prilagođavanje težina za neku funkciju, osim nasumičnih prilagodbi za čije se promjene rezultati ne bi bili jasno vidljivi. Također je potrebno napomenuti kako točno vrijednost pogreške utječe na prilagođavanje težina funkcije. Naime, kako bi se dobila „vrijednost kojom će se prilagoditi postojeće težine nekog neurona potrebno je pomnožiti stopu učenja zajedno s iznosom pogreške i vrijednosti izlaza kojeg je taj neuron proslijedio dalje u mrežu“ [11, p. 276]. Iz toga je jednostavno primijetiti kako u slučajevima kada je iznos pogreške što bliže nuli, to se manje promjene vrše na težine tog neurona, jer odstupanja između stvarnog i traženog izlaza isto postaju sve manja. „Najčešća mjera uspjeha za treniranje ANN-a na način nadziranog učenja je kvadratna

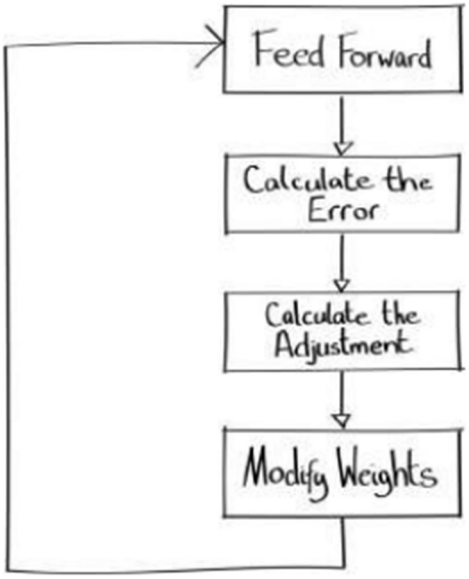
euklidska udaljenost, to jest greška između vektora stvarnog izlaza ANN-a (α) i traženog izlaza ***y***, koja se dobije sa sljedećom formulom: $E = \frac{1}{2}\sum_j \left(y_j - \alpha_j\right)^2$ gdje se suma sastoji od pojedinačnih razlika u izlazima svih izlaznih neurona, to jest neurona koji se nalaze u izlaznom sloju. Također je potrebno napomenuti da su y_j oznake konstantne vrijednosti te da je E funkcija svih težina ANN-a pošto stvarni izlazi ovise o njima“ [7, p. 62].

Algoritam povratnog rasprostiranja pogreške

Algoritam povratnog rasprostiranja pogreške se temelji na „pravljenju malih promjena kao odgovora na svaku grešku koju mreža napravi i slanju tih ispravaka nazad kroz ANN, krećući se od izlaznog sloja, pa skroz do ulaznog“ [10, p. 66]. Također je potrebno istaknuti kako „algoritam povratnog rasprostiranja implementira gradijent spusta za optimizaciju, to jest minimiziranje funkcije pogreške“ [6], te je najpopularniji algoritam koji se koristi za treniranje ANN-ova [7]. Radi preglednosti tijekom čitanja, odsad u ovome poglavlju će se za algoritam povratnog rasprostiranja koristiti njegov engleski naziv backpropagation. Backpropagation algoritam je dobio svoje ime po tome što se promjene kreću od kraja ANN-a, to jest njegovog izlaznog sloja, skroz kroz njegove skrivene slojeve, dok ne dođu do ulaznog sloja. To je algoritam čiji je zadatak ažurirati težine svakog neurona za malu količinu tijekom kretanja od izlaznog do ulaznog sloja, kako bi se izlaz ANN-a približio traženome izlazu što je više moguće [11]. Dakle, backpropagation mora „izračunati parcijalnu derivaciju, to jest gradijent funkcije pogreške E za svaku težinu neurona unutar ANN-a i prilagoditi te težine neurona ANN-a prateći suprotni smjer od gradijenta koji minimizira E funkciju pogreške“ [7, p. 63]. Prema gore danoj definiciji zadatka algoritma backpropagation, moguće je zaključiti kako taj algoritam zapravo služi samo kao metoda za izračunavanje gradijenta, „dok je potrebno koristiti se nekim drugim algoritmom, poput algoritma stohastičkog gradijenta spusta, za učenje ANN-a uz pomoć izračunatog gradijenta“ [4, p. 204]. Algoritam stohastičkog gradijenta spusta se također koristi za računanje dobivenog gradijenta u slučaju projekta MicroMachine.AI autora Cédric Bovar [9].

Zato što izlaz ANN-a α ovisi i o parametrima težina ***w*** i ***b***, a ne samo o vrijednosti ulaza ***x***, postaje moguće izračunati gradijent E za bilo koju težinu veze među neuronima ***w*** i težinu pristranosti nekog neurona koji se nalazi u ANN-u ***b***, što će odrediti koliko će promjena tih težina utjecati na vrijednost funkcije pogreške. Naravno da je potrebno imati neki način mjerenja poželjnosti takvih promjena, tako da se za to koristi parametar $\eta \in [0, 1]$ koji se zove stopa učenja [7]. „U slučaju kada nema informacija o općenitom obliku funkcije između pogreške i težina, ali postoje informacije o njezinom gradijentu, korištenje pristupa korištenog u algoritmu spusta gradijenta je prikladno za pronalazak globalnog minimuma E funkcije“ [7, p. 63]. Kao što je definirano u prethodnom poglavlju, pristup spusta gradijenta bi u ovome

slučaju započeo odabirom neke nasumične točke koja se nalazi na gradijentu i pripada prostoru težina. Od te točke bi se krenuo pratiti gradijent najstrmijim nagibom prema dolje u smjeru nižih vrijednosti E funkcije, ako ne i globalnog minimuma funkcije. Kao i u tipičnome slučaju uporabe algoritma gradijenta spusta, ovaj proces se ponavlja iterativno skroz dok se ne dostignu željene vrijednosti E, od kojih je najtraženiji globalni minimum, ali tipično je doseći neki lokalni minimum funkcije E koji je dovoljno nisko da ga se prihvati ili u drugom slučaju gdje se proces ponavlja dok se ne potroše dostupni računski resursi.



Slika 8: Primjer procesa algoritma povratnog rasprostiranja pogreški (*engl. backpropagation*) *[11]*

Kao što je prikazano na slici broj 8 backpropagation proces se može podijeliti na petlju sastavljenu od četiri koraka, ali ti koraci ne opisuju ni pripremu petlje niti izlazak iz nje. Kako bi se pripremio algoritam za petlju, „potrebna je inicijalizacija parametara težine neurona ***w*** i ***b*** kao nasumičnih i uobičajeno malih vrijednosti“ [7, p. 63]. Nakon ulaza u petlju, svaki par ulaza i izlaza koji se koristi za treniranje, ponajprije mora proći kroz normalni proces funkcioniranja ANN-a koji radi prema naprijed. To znači da mora pružiti uzorak ulaza ***x*** koji će proći kroz ANN te nakon potrebnog računanja izaći kao stvarni izlaz tog ANN-a α_j za taj par ulaza i izlaza [7]. Sljedeća dva koraka su relativno jednostavna u tome da je samo potrebno pratiti formule koje su već navedene i objašnjene u ovome poglavlju. Prvo se treba izračunati funkcija pogreške za korišteni par ulaza i izlaza, te onda iskoristiti dobivena vrijednost E kako bi se mogla izračunati derivacija pogreške za svaku težinu veze neurona i težinu pristranosti neurona unutar ANN-a, krećući se od izlaznog prema ulaznome sloju. Kao zadnji korak preostaje samo ažurirati novo dobivene težine te u slučaju da E još nije dovoljno malen nastaviti petlju, naravno samo ako je ostalo dovoljno računskih resursa za nastavljanje njenog rada [7].

Ograničenja i rješenja

Bitno je napomenuti da zato što se backpropagation algoritam temelji na algoritmu optimizacije gradijentu spusta, dijeli probleme koji zahvaćaju taj algoritam. Dakle, ne može se očekivati da će backpropagation algoritam uvijek moći pronaći globalni minimum od funkcije E zato što kao i gradijent spusta, može imati problema s prijelazom preko lokalnih minimuma koji se nalaze prije globalnog minimuma na krivulji. „Zbog toga što su to područja gdje je vrijednost gradijenta blizu nuli, prolazak kroz njih rezultira u ažuriranjima težina koje su blizu nuli i koja dovode do preuranjene konvergencije algoritma“ [7, p. 64]. Uobičajena rješenja i dodatci algoritmu za savladavanje konvergencije do lokalnog minimuma uključuju:

- **Nasumična ponovna pokretanja:** „Algoritam se može ponovno pokrenuti s novim nasumičnim vrijednostima težine u nadi da ANN nije previše ovisan o sreći. Nijedan ANN model nije dobar, ako previše ovisi o sreći; primjer bi bio da dobro radi u samo jednom ili dvaput u deset pokušaja“ [7, p. 64].
- **Dinamična stopa učenja:** Može se promijeniti parametar stope učenja i promatrati razlike u performansama ANN-a ili uvesti parametar dinamičke stope učenja koji se povećava kada je konvergencija spora i smanjuje kada je konvergencija do nižih vrijednosti E brža [15].
- **Inercija:** Kao alternativa se također može dodati količina inercije ažuriranju težina kao: Nova promjena težine je jednaka umnošku parametra inercije i stare promjena težine umanjenog za korak učenja. U navedenoj formuli parametar inercije je broj između $[0,1]$, a korak učenja se računa kao težina pomnožena sa stopom učenja. Dodatak vrijednosti inercije prethodnom ažuriranja težine može pomoći backpropagation algoritmu da savlada potencijalne lokalne minimume, kao što to čini za gradijent spusta [16].

Dok se gore navedena rješenja mogu koristiti za ANN-ove manje veličine, dokazano je na praktičnim primjerima kako se većina ovih problema ne pojavljuju u modernim, dubokim ANN arhitekturama, što sugerira da su navedena ograničenja većinom eliminirana u tim slučajevima [17].

Grupno naspram pojedinačnog treniranja

Backpropagation algoritam je moguće primijeniti na neki ANN nakon što je on prošao grupno (*engl. batch*) ili pojedinačno (*engl.non-batch*) treniranje. U pojedinačnom treniranju dolazi do ažuriranja težina veza među neurona i težina pristranosti svaki put kada se novi primjerak za treniranje predstavi ANN-u, a provedeno ažuriranje se temelji na razlici između izračunatih, to jest stvarnih izlaza ANN-a te traženih izlaza za koje se zna da odgovaraju pruženom ulazu. Dok se u grupnom treniranju težine ažuriraju tek nakon što su svi primjerci

za treniranje predstavljeni ANN-u tijekom čega se akumuliraju promjene težine veza i težine pristranosti pojedinačnih neurona za svaki primjerak, a provedeno ažuriranje se temelji od primjenjivanja akumuliranih težina na težine neurona svakog pojedinačnog primjerka. [18] „Individualni oblik je nestabilniji jer kroz iteracije ovisi o jednoj podatkovnoj točki, iako to može biti korisno u izbjegavanju nepoželjnih konvergencija do lokalnog minimuma. S druge strane, grupni oblik je stabilniji pristup spusta gradijenta jer ažuriranje težina ovisi o prosječnoj pogreški svih primjeraka za treniranje u grupi“ [7]. Najbolji način za iskorištavanje prednosti oba pristupa je korištenjem grupnog učenja s nasumično odabranim primjercima u grupama male veličine, to jest korištenjem takozvanog „mini-serija“ (*engl. mini-batch*) treniranja [6]. Dakle, najbolja opcija je mini-serija treniranje, jer je on znatno brži i stabilniji od individualnog oblika, dok uspijeva izbjeći problem generalizacije time što nema samo jednu prosječnu pogrešku za sve primjerke u skupu primjeraka za treniranje kao što je slučaj u grupnom obliku treniranja.

4.2.1.5. Tipovi ANN-a

„Osim standardnih feed forward MLP-a, postoje brojni tipovi ANN-a koji se koriste za klasifikaciju, regresiju, učenje sklonosti, procesiranje i filtriranje podataka i raspoređivanje zadataka u razrede“ [7, p. 65]. Također je bitno napomenuti da postoje i povratne neuronske mreže, koje se mogu objasniti kao ANN-ovi koji su prošireni da uključuje povratne veze i gdje je dopušteno vezama između neurona da stvore usmjerene cikluse, što omogućava ANN-u da uhvati dinamičke i vremenske fenomene, to jest fenomene koje je lako predstaviti kao sekvence podataka [4]. Postoji još tipova ANN-a, ali oni nisu spomenuti u ovome radu.

4.2.1.6. Od plitkih do dubokih ANN-a

Jedan od ključnih parametara za treniranje ANN-a je njegova veličina, pod veličinom se misli na dubinu ANN-a, što je osobina koja je pojašnjena prije u poglavlju, ali čija će definicija biti ponovljena ovdje. „Općenito je prihvaćeno pravilo koje predlaže da veličina neuronske mreže treba odgovarati kompleksnosti problema koji se pokušava riješiti“ [7, p. 65]. Dubina ANN-a je određena brojem skrivenih slojeva koje ta mreža sadrži. Ian Goodfellow i ostali tvrde sljedeće: „Zbog terminologije korištene kod opisa broja skrivenih slojeva mreže dolazi se do imena duboko učenje“ [4]. Na temelju toga moguće je zaključiti kako ANN arhitekture koje sadržavaju barem jedan skriveni sloj se mogu nazvati dubokim mrežama, dok se slojevite arhitekture koje imaju samo jedan izlaz, ali bez skrivenih slojeva mogu nazvati plitkima [7].

5. Učenje s hibridnim algoritmom

Slično kao i u prirodi, metode za treniranje modela umjetne inteligencije je moguće kombinirati na brojne načine kako bi se dobili novi sofisticiraniji hibridni algoritmi, gdje suma metoda postaje bolja od pojedinačnih metoda [7]. Rezultati kombinacija metoda umjetne inteligencije koje nastanu na ovaj način se nazivaju hibridnim algoritmima i u ovom radu je pružen detaljniji pogled u jedan od „najviše obećavajućih hibridnih algoritama za rješavanje problema strojnog učenja neuronskih mreža“ [19, p. 1]. Hibridni algoritam o kojemu je riječ je neuroevolucija, također poznata pod imenom evolucija umjetnih neuronskih mreža. Većina poglavlja je posvećena NEAT (*engl. NeuroEvolution of Augmenting Topologies*) algoritmu, koji se temelji na neuroevoluciji, zato što se taj algoritam koristi u prvome analiziranome projektu Watchcarslearn autora Manas Sarpatwar [5].

5.1. Neuroevolucija

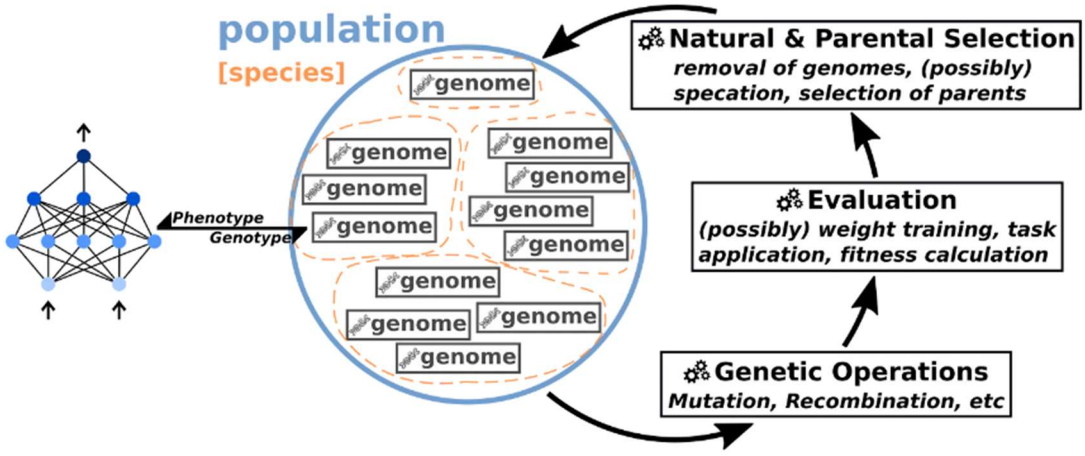
Evolucija umjetnih neuronskih mreža ili neuroevolucija, se odnosi na „umjetnu evoluciju neuronskih mreža koristeći se genetskim algoritmima“ [20, p. 655]. Tu tvrdnju potvrđuju i Georgios N. Yannakakis te Julian Togelius, kada govore kako se „neuroevolucija odnosi na dizajniranje umjetnih neuronskih mreža, dakle težinu veza između neurona u toj neuronskoj mreži, topologiju te mreže ili oboje, koristeći se evolucijskim algoritmima“ [7, p. 83] u svojoj knjizi Artificial Intelligence and Games. „Neuroevolucija pretražuje prostor ponašanja kako bi pronašla neuronsku mrežu koja postiže dobre rezultate za traženi problem“ [21, p. 99], pod prostorom ponašanja se misli na skup ANN-ova koji zbog svojih različitih težina i topologija imaju drukčije rezultate, pa tako i drukčije razine uspješnosti tijekom rješavanja traženog problema. Imajući na umu dane izjave, a i drugi naziv za neuroevoluciju, postaje logično postaviti pretpostavku da je neuroevolucija jednostavno algoritam u kojemu ANN-ovi prolaze evoluciju skroz dok se evolucijom ne dobije ANN koji ostvaruje zadovoljavajuću razinu uspješnosti u rješavanju zadanog problema. Isto tako se može pretpostaviti kako se evolucija događa promjenama u težinama veza među neuronima ANN-a ili topologiji ANN-a, koje se dobiju korištenjem genetskih algoritama. „Neuroevolucija je jako obećavajući pristup za primjenu u domenama prilagodljivog upravljanja, evolucijske robotike, strategija korištenih u računalnim igrama, sustava s više agenata i drugim domenama u kojima bi bilo nemoguće koristiti se metodama i algoritmima nadziranog učenja“ [19, p. 1]. Postoje dva razloga zbog kojih je moguće primijeniti ovaj algoritam za toliko različitih problema. Prvi od razloga je taj što se „mnogo problema koji zahvaćaju umjetne inteligencije može promatrati kao funkciju problema optimizacije, što znači da ju je moguće aproksimirati koristeći se ANN-ovima“ [7, p.

83], zato što kao što je ranije objašnjeno, ANN-ovi se koriste backpropagation algoritmom koji se temelji na algoritmu optimizacije, gradijentu spusta. Drugi razlog je što je neuroevolucija metoda koja se temelji na biologijskim metaforama i teoriji evolucije te joj je kao velika inspiracija poslužio biološki proces evolucije mozgova u prirodi [22].

„Primjena ovakvog pristup evolucijskog učenja se događa ili u slučajevima kada dostupna funkcija pogreške nije diferencijabilna ili u slučajevima gdje traženi izlazi nisu dostupni. Jedan od mogućih primjera prvog slučaja je kada aktivacijske funkcije korištene u umjetnoj neuronskoj mreži nisu kontinuirane i stoga nisu diferencijabilne“ [7, p. 83]. To je poznat fenomen u mrežama koje proizvode kompozicijske uzorke (eng. *Compositional Pattern Producing Networks - CPPN*) što one imaju povećani broj aktivacijskih funkcija, neke od kojih nisu diferencijabilne što sprečava korištenje backpropagation algoritma [23]. Drugi slučaj se može pojaviti u domeni za koju nema primjeraka dobrog ili lošeg ponašanja, to jest ponašanja koje bi doprinosilo uspješnosti ANN-a i ponašanja koje bi tu uspješnost smanjivalo. Također je potrebno spomenuti situacije u kojima jednostavno nije moguće jasno i objektivno definirati što bi to točno bilo dobro, a što loše ponašanje ANN-a [7]. Umjesto korištenja backpropagation metode za računanje funkcije pogreške i provođenje ažuriranja ANN-a ovisno o algoritmu pretraživanju gradijenta, poput algoritma stohastičkog gradijenta spusta, neuroevolucija dizajnira strukture ANN-a koristeći se evolucijskom pretragom [22]. Za razliku od nadziranog učenja, za učenje uz pomoć neuroevolucije nije potreban početni skup podataka, koji se sastoji od parova ulaza i izlaza, za treniranje ANN-a [19]. „Neuroevolucija zahtijeva samo neko mjerilo performanse ANN-a koje će jasno prikazati uspješnost ANN-a u rješavanju zadanog problema“ [7, p. 83]. Funkcija koja mjeri tu uspješnost se također može zvati i funkcijom prikladnosti.

Neuroevolucija je metoda hibridnog učenja koja generira sve bolje neuronske mreže za rješavanje problema na koji se primjenjuje, dok se ne ostvari zadovoljavajuće prikladna neuronska mreža ili se ne potroše računski resursi [24]. Moguće je taj proces učenja podijeliti na sedam temeljnih algoritamskih koraka [5]. Jedino se ne smije zaboraviti napomenuti da se u stvari radi o jednom koraku pripreme, petlje učenja koja se sastoji od pet koraka, te dodatnog koraka koji služi samo kako bi se izašlo iz petlje. 1) Prvi korak, to jest priprema za početak petlje, se sastoji od „evoluiranja populacije kromosoma koji predstavljaju ANN-ove, kako bi se optimizirala funkcija prikladnosti koja karakterizira kvalitetu reprezentacije ANN-ova. Tipično je da se populacija kromosoma inicijalizira nasumično“ [7, p. 83]. Kromosomi se u tom kontekstu mogu poistovjetiti s genomima koji se definiraju kao „pojednostavljene genetske reprezentacije neuronskih mreža, to jest ANN-ova“ [24], svaka sa svojim odgovarajućim težinama i topologijom. Sljedećih pet koraka svi spadaju u petlju učenja neuroevolucije, počevši od 2) kodiranja svakog ANN u njegov odgovarajući genom, te testiranja tog ANN-a na zadatku koji prolazi optimizaciju. 3) Nakon testiranja, svakom genomu unutar populacije se dodjeljuje

njegova vrijednost prikladnosti. Vrijednost prikladnosti nekog genoma se koristi kao mjerilo performansi odgovarajućeg ANN-a tijekom testiranja zadatka [24]. 4) „Nakon što je određena vrijednost prikladnosti za sve kromosome u trenutnoj populaciji, potrebno je odabrati strategiju koja će se koristiti za određivanje roditelja sljedeće generacije“ [7, p. 83]. Slijedi 5) stvaranje nove populacije potomaka primjenom genetskih operatora nad odabranim genomima. S primjenom mutacija se potiče korištenje novih težina i struktura ANN-a, dok se križanjem između odabranih genoma šire poželjne osobine na ostatak populacije [11]. Konačni korak petlje se sastoji od 6) „primjene strategije zamjene kako bi se odredili posljednji članovi nove populacije“ [7, p. 83]. I kao sedmi korak se logično 7) provjerava da li su potrošeni računski resursi ili je ostvarena zadovoljavajuća razina prikladnosti trenutne populacije kako bi se utvrdilo da li se izlazi iz petlje.



Slika 9: Primjer tipičnog generacijskog procesa neuroevolucije*[24]*

Dakle, ako se uzmu u obzir dosad iznesene informacije zajedno sa slikom broj 9, može se zaključiti kako neuroevoluciji nije potreban skup početnih podataka za treniranje zato što algoritam sam nasumično inicijalizira skup ANN-ova prije kretanja u petlju. Ti ANN-ovi su predstavljeni uz pomoć genoma, kompaktnih reprezentacija težina i topologije nekog ANN-a. Nakon testiranja ANN-ova u skupu, to jest populaciji, ako je neki od njih vrlo uspješan, genom tog ANN-a će se križati s genomima iz drugih uspješnih ANN-ova te tako proširiti svoje poželjne osobine na ostatak populacije. Za razliku od njih, neuspješne ANN-ove i njihove genome će se ili odbaciti ili će se nad njima provesti mutacije kako bi se dobili ANN-ovi s novim strukturama, težinama ili potpuno promijenjeni ANN-ovi koji imaju drugačiju ne samo strukturu već i težine ANN-a. Naravno, točno koji će se genomi križati s kojima i koji će mutirati ovisi o odabranoj strategiji određivanja. Algoritam neuroevolucije ponavlja taj ciklus dok se ne potroše dostupni resursi ili se ne dosegne razina uspješnosti koja je zadovoljavajuća.

Evoluciji mreža u neuroevoluciji se može pristupiti na klasični način, ili se koristiti i evolucijom strukture mreža u populaciji [20]. Pod klasični tip pristupa spadaju oni koji se fokusiraju samo na evoluciju težina veza između neurona i težina pristranosti mreže, dok imaju fiksnu topologiju mreže. Drugi pristupi su oni koji osim što provode evolucije težina mreže, također evoluiraju i topologiju mreže, uključujući tipove veza i aktivacijske funkcije [21]. U prvom tipu pristupa „vektor težine je kodiran i reprezentira se genetički kao genom“ [7, p. 84], dok u drugom tipu „genetska reprezentacija sadržava i kodiranu verziju topologije ANN-a“ [7, p. 84]. Također je potrebno napomenuti kako u prvom pristupu se „od korisnika očekuje da odredi prikladnu strukturu i broj neurona u skrivenim slojevima unutar ANN-a, prije nego je moguće krenuti raditi s njime. Što je važno jer topologija neke mreže znatno utječe na njene performanse“ [25, p. 8]. Prema tome je moguće reći kako je glavna razlika između ta dva tipa u tome da li je topologiju potrebno odrediti ručno prije početka učenja te se ne mijenja kroz generacije ili ona nasumično određena u početku te prolazi kroz proces evolucije pa se razvija i minimizira tijekom generacija.

Neuroevolucija je široko primijenjena u domeni igara u ulogama poput procjenjivanja prostora stanja-radnje u igri, odabira ispravne radnje, odabira između mogućih strategija, stvaranja modela protivničkih strategija, kreiranja sadržaja i modeliranja iskustva igrača [25]. Efikasnost, skalabilnost, mogućnost široke primjene, sposobnost učenja koja nije ograničena s kompleksnosti problema, pamćenja prošlih stanja agenta, računanja proizvoljno kompleksnih funkcija i sposobnost generalizacije ponašanja za prije neviđene ulaze su samo neki od razloga koji čine neuroevoluciju općenito dobrom metodom za obavljanje brojnih zadataka koji bi bili zadani agentu s umjetnom inteligencijom u nekoj igri [20].

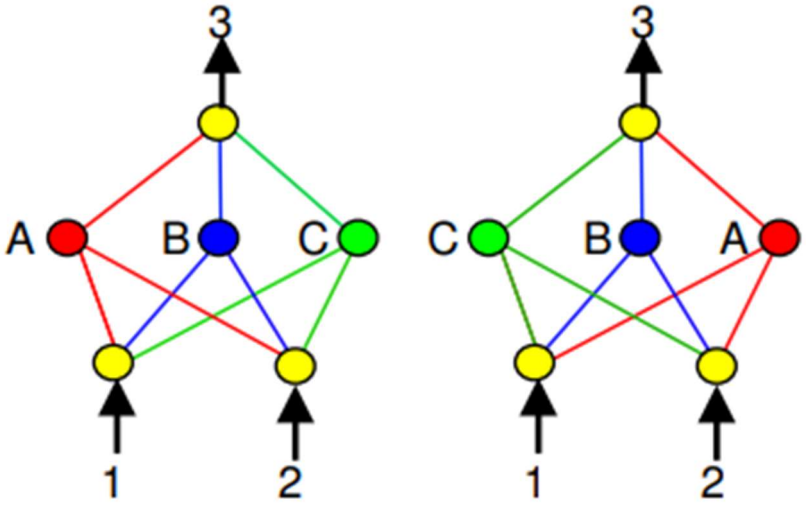
5.1.1. Neuroevolucija augmentirajućih topologija

Algoritam neuroevolucije augmentirajućih topologija (*engl. NeuroEvolution of Augmenting Topologies - NEAT*) je kreiran kako bi se riješili problemi koji su nastali tijekom neuroevolucije poput provođenja procesa križanja između dviju mreža s različitim topologijama, zaštite kompleksnih struktura od preuranjenog izumiranja kako bi se dosegnuo njihov pun potencijal nakon optimizacije te minimiziranja strukture ANN-a kako bi se dosegnula najefikasnija rješenja [22]. NEAT algoritam spaja uobičajenu evoluciju težina veza između neurona i težina pristranosti mreže zajedno s augmentirajućom topologijom [21]; što omogućava da se s „kombinacijom tradicionalne potrage za težinama ANN-a i novim razvijanjem kompleksnosti njegove strukture, ostvari sofisticiranije ponašanje evoluiranih mreža tijekom generacija“ [20, p. 655]. Zato što „ANN-ovi korišteni unutar NEAT algoritma nastaju evolucijom ne samo težina mreže već i evolucijom njezine topologije, takve mreže se nazivaju umjetnim neuronskim mrežama koje evoluiraju topologiju i težine mreže (eng.

Topology and Weight Evolving Artificial Neural Networks - TWEANNs)“ [19, p. 1] ili koristeći se engleskom kraticom TWEANN-ovima. U ovome su poglavlju objašnjena tri problema neuroevolucije i rješenja kojima se NEAT algoritam koristi kako bi ih izbjegao. Također će biti pobliže objašnjeno genetičko kodiranje koje se koristi u NEAT algoritmu kako bi neki od problema i rješenja bili jednostavniji za shvatiti. „NEAT algoritam je jedinstven po tome što njegove strukture postaju sve kompliciranije što su optimiziranije, što samo pojačava simboličnu povezanost između genetskih algoritama i prirodne evolucije“ [21, p. 101].

5.1.1.1. Problem konkurentnih konvencija

Jedan od glavnih problema neuroevolucije je problem konkurentnih konvencija (eng. *Competing Conventions Problem*) [26]. Ovaj problem se može pojaviti tijekom procesa križanja genoma koji su odabrani da budu roditelji novoj generaciji potomaka [24]. Naime, do konkretnog problema dolazi kada se „križaju genomi koji nisu kodirani na isti način, jer postoji velika šansa za stvaranje oštećenih potomaka“ [21, p. 103], što znači da može doći do gubitka informacija o nekim osobinama ANN-ova koji su bili kodirani u te genome. Problem je dobio ime po tome što ga uzrokuju konkurentne konvencije, zato što postoji više od jednog načina, ili konvencije za izražavanje rješenja problema optimizacije težina koristeći se neuronskom mrežom, što onda uzrokuje ovaj problem kada su za dva različita ANN-a korištene dvije različite konvencije za izražavanje rješenja [21]. Primjer problema konkurentnih konvencija je vidljiv na slici broj 9.



Slika 10: Primjer problema konkurentnih konvencija *[27]*

Na slici broj 10 je prikazan primjer problema konkurentnih konvencija u kojemu dva različita ANN-a računaju istu funkciju. „Problem je u tome što se neuroni A, B i C koji se nalaze u njihovim skrivenim slojevima pojavljuju u drugačijim redoslijedima i predstavljeni su uz

pomoć različitih kromosoma, što ih čini nekompatibilnima za križanje“ [27, p. 18]. U slučaju križanja, korištenjem operatorom jedne točke križanja (*engl. Single Point Crossover Operator*) [28], između navedenih primjera (A, B, C) i (C, B, A) s točkom križanja B bi rezultiralo u gubitku jednog od tri glavna skrivena neurona kromosoma te informacija koje je on posjedovao. To je zato što bi novonastali potomci mogli biti ili (A, B, A) ili (C, B, C). Dakle, dolazi do problema, jer potomci gube cijelu trećinu informacija koje su im roditelji posjedovali. „Općenito za n skrivenih neurona postoji $n!$ funkcijski ekvivalentnih rješenja“ [21, p. 103], to jest moguće ih je prikazati na $n!$ načina.

Postoji više razloga zbog kojih može doći do problema konkurentnih konvencija, naime za taj problem je samo bitno da se reprezentacije genoma ne poklapaju. To je moguće ostvariti razlikama u veličini genoma ili razlikama u smještaju gena na pozicijama unutar genoma. Dakle, ako geni koji su na istim pozicijama ne predstavljaju iste osobine za oba genoma ili geni koji izražavaju iste osobine u oba genoma, ali se ne nalaze na istim pozicijama unutar njih [21]. NEAT algoritam rješava ovaj problem korištenjem povijesnih oznaka kako bi algoritam bio siguran koji je gen koji tijekom učenja [20], ali pojašnjenje tog načina zaštite će biti pruženo kasnije u ovome radu.

5.1.1.2. Problem zaštite inovacije sa specijacijom

„Kada se govori o inovaciji u TWEANN-ovima, ona se ostvaruje dodavanjem nove strukture mrežama kroz proces mutacije postojeće strukture“ [27, p. 24]. Kao i u stvarnome svijetu, ponekad nove strukture koje se pojave u mreži nisu istog trena korisne, već često smanjuju prikladnost mreže. Štoviše, vrlo je vjerojatno da neki novi čvor ili veza neće biti korisne za rješavanje traženog zadatka odmah nakon što su dodani, već im je potrebno pružiti nekoliko generacija vremena kako bi se nova struktura optimizirala i pravilno iskoristila [20]. Nažalost, ako ne bude zaštićena, najvjerojatnije je da bilo kakva inovacija ne poživi dovoljno dugo da postane optimizirana. To se događa zato što njen početni negativni utjecaj na prikladnost mreže u kojoj se nalazi znači da će ju se vrlo vjerojatno izbaciti prilikom ulaza u sljedeću generaciju. Stoga je potrebno pružiti zaštitu mrežama s inovativnim strukturama kako bi imale priliku optimizirati i potpuno iskoristiti svoje nove strukture [21].

Kako bi se zaštitio od ovog problema, NEAT algoritam je inspiraciju pronašao u prirodi i vrstama koje postoje u njoj. Naime, „u prirodi različite strukture tipično pripadaju različitim vrstama koje se natječu u različitim nišama. S time se implicitno štite inovacije unutar niša“ [27, p. 25]. Imajući to na umu može se zaključiti da ako se mreže s inovativnim strukturama izolira u vlastite vrste, onda bi te mreže imale priliku optimizirati svoje strukture bez opasnosti preuranjenog nestanka iz populacije. Time bi takve mreže imale priliku natjecati se sa ostatkom populacije tek nakon što dostignu svoj puni potencijal, što bi eliminiralo nepotrebni gubitak

inovativnih struktura koje mogu biti ključne za rješavanje zadatka, ali imaju negativni početni utjecaj na prikladnost mreže [21].

„Specijacija je proces kojim se genomi populacije grupiraju prema sličnostima nekih njihovih osobina, poput sličnosti u topologiji ili ostvarivanja određene razine uspjeha u rješavanju zadatka“ [24]. Kako bi se mogla provoditi specijacija potrebno je moći razlikovati da li neka dva genoma pripadaju istoj vrsti ili ne. „Taj zadatak je dodatno otežan time što problem konkurentnih konvencija uzrokuje da mreže koje računaju istu funkciju izgledaju vrlo različito“ [21, p. 105]. Povijesne oznake kojima se NEAT algoritam koristi kako bi riješio problem konkurentnih konvencija usput „omogućuju sustavu da podijeli populaciju na vrste ovisno o sličnosti njihovih topologija“ [20, p. 657]. Koristi se eksplicitno dijeljenje prikladnosti, kako bi pojedinačni genomi unutar vrsta dijelili vrijednost prikladnosti sa ostatkom genoma koji su u istoj vrsti s njima [27]. Eksplicitna verzija je prikladna za TWEANN-ove jer dopušta grupiranje mreža na temelju topologije i konfiguracija težina, dok odgovara NEAT algoritmu jer je jednostavno izmjeriti sličnost između dva genoma, koristeći se povijesnim oznakama koje se nalaze u njihovim genima. „Rezultat dijeljenja prikladnosti je to što broj mreža koje mogu postojati u populaciji na jednom vrhu prikladnosti su ograničene veličinom tog vrha.“ [21, p. 105]. Dakle, logično je zaključiti kako se onda populacija podijeli na broj vrsta, svaka od kojih ima vlastiti vrh, „što sprečava ijednu vrstu od prevladavanja ukupnom populacijom“ [20, p. 657]. „Eksplicitno dijeljenje prikladnosti je pogodno NEAT-u jer se sličnost može jednostavno izmjeriti na temelju povijesnih oznaka koje su pohranjene u genima“ [21, p. 105].

5.1.1.3. Problem inicijalne populacije i topologijske inovacije

Mnoge TWEANN metode se koriste inicijalnim populacijama koje imaju nasumično odabrane topologije, kako bi osigurali raznolikost od početka treniranja [27]. Probleme u takvom pristupu uzrokuju slučajevi poput „kreiranja mreža u kojima nije moguće pronaći put od svakog ulaza do svakog izlaza neke mreže“ [21, p. 105], koji onda zahtijevaju potrošnju dodatnog vremena kako bi se riješili.

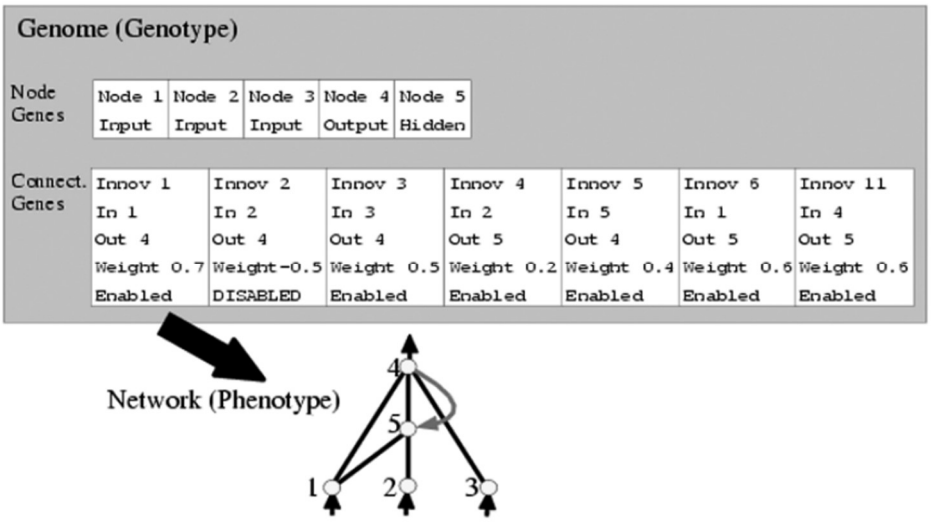
Kao što su potvrdili u svojim eksperimentima, Stanley i Miikkulainen u svome radu Evolving Neural Networks through Augmenting Topologies govore kako je „poželjno evoluirati minimalna rješenja zato što se tako dolazi do manjeg broja parametara koji se trebaju pretraživati“ [21, p. 105]. „Početak s nasumičnim topologijama ne vodi do pronalaska minimalnih rješenja“ [19, p. 1] jer u tome slučaju populacija od početka ima puno nepotrebnih nasumičnih čvorova i veza koje se moraju ukloniti kako bi se mreža mogla početi minimizirati. Rješenje koje koristi NEAT algoritam je da minimizira početnu strukturu svih mreža te ih postupno razvija kroz generacije [27]. Zato njegova populacija započne bez skrivenih čvorova

i razvija svoju strukturu samo kada je to korisno rješenju, tako da je jedina razlika između početnih mreža ona u njihovim inicijalnim nasumičnim težinama [20].

„Time što započinje minimalno, NEAT osigurava da će sistem tražiti rješenja u prostoru težina koji ima najmanje moguće dimenzija tijekom svih generacija“ [21, p. 106]. Time se ujedno „znatno smanjuje potreban broj generacija za pronalazak rješenja te osigurava da mreže neće postati kompliciranije nego što je potrebno“ [20, p. 658]. Dakle, samim time što je njegova početna struktura minimalna, NEAT algoritam poboljšava šanse pronalaska globalnog optimuma i skraćuje vrijeme potrebno za to, dok istovremeno osigurava minimiziranje mreža korištenih za dobitak tog rješenja. Sve to rezultira dramatičnim poboljšanjima performansi NEAT algoritma, te je jedan od razloga zbog kojeg je NEAT visoko efektivan i ima prednost nad drugim metodama neuroevolucije koje se koriste fiksnim topologijama na području računalnih igara [20].

5.1.1.4. Genetsko kodiranje

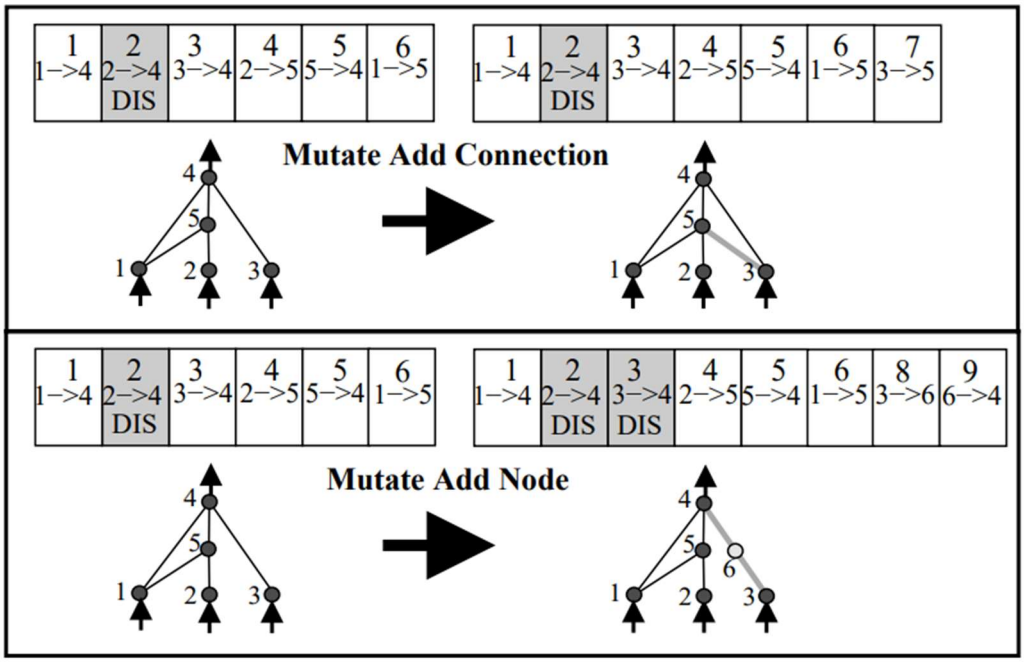
Genetsko kodiranje omogućuje NEAT algoritmu da brzo i jednostavno analizira informacije o neuronskim mrežama koje se nalaze u njegovoj populaciji. Čini to tako što predstavlja inače kompleksne podatkovne strukture ANN-a kao kompaktne genetske kodove koje je moguće puno brže procesirati [24]. Naravno, ne smije se zaboraviti da „kako bi se strukture mreža u NEAT algoritmu mogle razvijati u kompleksnosti potrebno je da njihove reprezentacije budu dinamične i da ih je moguće proširiti“ [27, p. 34]. Zato je također shema za genetsko kodiranje u NEAT-u „dizajnirana da dopušta jednostavno poravnavanje odgovarajućih gena kod križanja genoma tijekom reprodukcije“ [21, p. 107]. „Genomi su pojednostavljene genetičke reprezentacije neuronskih mreža“ [24], kao što je prikazano na slici broj 10. Imajući na umu iznad navedene informacije, može se reći kako se genomi u NEAT algoritmu koriste zato da bi se ubrzao proces dolaska do rješenja tako što sustav ne mora analizirati jedinke u populaciji kao kompleksne neuronske mreže, već ih može jednostavno procesirati kao njihovu kompaktnu genetičku reprezentaciju. „Svaki genom sadrži listu veznih gena, svaki od kojih upućuje na dva gena čvora s kojima je on povezan. Geni čvora pružaju listu ulaza, skrivenih čvorova i izlaza koji se mogu spojiti. Svaki gen veze određuje ulazni čvor, izlazni čvor, težinu veze, da li je gen veze izražen, i broj inovacije, koji dopušta pronalazak odgovarajućih gena tijekom procesa križanja“ [27, p. 34]. Dakle, svaki gen čvora koji se nalazi unutar genoma sadrži informacije o tipu neurona kojeg reprezentira i u kojem se sloju taj neuron nalazi, bilo to ulaznom, skrivenom ili izlaznom. Dok svaki vezni gen sadrži informacije o težini veze kojom utječe na vrijednosti koje prolaze od neurona gdje veza počinje do neurona gdje ona završava. Također sadržava redoslijedne brojeve tih neurona, informaciju o statusu izraženosti tog gena i inovacijski broj veze.



Slika 11: Primjer mapiranja genotipa u fenotip *[20]*

Slika broj 11 prikazuje „genom, također zvan genotipom, i neuronsku mrežu koju on reprezentira, koja se također naziva fenotipom. Prisutno je troje ulaznih čvorova, jedan skriveni čvor i jedan izlazni čvor, zajedno sa sedmero definicija veza među tih čvorova, jedna od kojih je povratna. Kao što je vidljivo u genotipu, drugi vezni gen je onesposobljen pa njegova veza (između čvorova 2 i 4) nije izražena u fenotipu“ [20, p. 656]. Mutacija u NEAT-u može promijeniti i težine veza i strukture mreža. Dok „težine veza prolaze kroz proces mutacije isto kao i u bilo kojem drugom neuroevolucijskom sustavu, gdje je svaka težina veze uznemirena koristeći se fiksnom vjerojatnošću dodavanjem broja s plivajućim zarezom, odabranog iz jednolike raspodjele pozitivnih i negativnih vrijednosti“ [27, pp. 34-35]. Mutacije strukture se mogu dogoditi na dva načina, kao što je i vidljivo na slici broj 12. Svaka mutacija povećava veličinu genoma dodavanjem novih gena, što ekvivalentno povećava kompleksnost mreže dodavanjem novih veza ili veza i neurona u drugom slučaju [22]. „U mutaciji dodatak veze, jedan novi gen veze s nasumičnom težinom se dodaje i spaja s dva dotad nepovezana čvora“ [20, p. 655]. „U dodatak čvora mutaciji, postojeća veza se dijeli i novi čvor se postavlja na mjesto stare veze. Stara veza se isključuje i dvije nove veze se dodaju genomu. Nova veza koja vodi do novog čvora dobiva težinu 1, a nova veza koja vodi od novog čvora dobiva istu težinu kao što je imala stara veza prije nje“ [27, p. 35]. Ova metoda dodavanja čvorova je odabrana kako bi se minimizirao inicijalni rezultat mutacije. „Stara ponašanja ostaju prisutna i ne mijenjaju se kvalitativno, ali nova nelinearnost u vezi pomaže razraditi ta originalna ponašanja mreže“ [20, p. 656], „dok se novi čvorovi mogu odmah integrirati u mrežu bez uništavanja prethodne strukture ili ometanja njenog rada, nasuprot dodavanja nepotrebne strukture koja bi se morala evolucijom dodati u mrežu kasnije“ [21, p. 108]. Na ovaj način, zahvaljujući specijaciji, mreža ima vremena optimizirati i iskoristiti svoju novu strukturu. Dakle,

proces mutacije u NEAT algoritmu može zahvatiti i težine veza ANN-a i strukturu ANN-a. Naspram mutacija strukture ANN-a, mutiranje njegovih težina veza je relativno jednostavno, jer se sastoji samo od dodijele neke pozitivne ili negativne vrijednosti svakoj težini veze u ANN-u koja bude odabrana fiksnom vjerojatnošću. Mutacije strukture se dijele na mutaciju dodavanja veze strukturi ANN-a i mutaciju dodavanja čvora, to jest neurona, strukturi ANN-a. Mutacija dodavanja veze je poprilično jednostavna, u tome što samo dodaje vezu između dva dotad nepovezana čvora, dok mutacija dodavanja čvora siječe jednu već postojeću vezu i postavlja novi čvor na njeno mjesto. Također spaja dvije nove veze između novog čvora i čvorova koje je prijašnja veza spajala. Veza koja ide prema novom čvoru ima novu težinu vrijednosti jedan, dok veza koja ide od novog čvora zadržava staru težinu veze.



Slika 12: Dva tipa mutacije strukture u NEAT algoritmu *[21]*

Na slici broj 12 su prikazana dva primjera mutacija strukture u NEAT algoritmu. Oba tipa su prikazani „s genima veze neke mreže prikazanima iznad njihovih fenotipa. Gornji broj u svakom genu je broj inovacije tog gena“ [21, p. 107]. Broj inovacije je povijesna oznaka koja se koristi za određivanje originalnog pretka svakog gena, gdje novi geni dobivaju sve veće brojeve. Prilikom dodavanja novih gena tijekom mutacije, novi gen se dodaje na kraj genoma i određuje mu se sljedeći dostupni broj inovacije [21]. Također se treba napomenuti kako u mutaciji dodavanja novog čvora je potrebno i onesposobiti gen veze koja se siječe [20].

Kako prolaze kroz mutaciju genomi u NEAT algoritmu logično postaju sve veći s prolazom vremena. „Kao rezultat se dobiju genomi različitih veličina, katkad s različitim vezama na istim pozicijama“ [27, p. 35]. Najkompliciranija forma problema konkurentnih

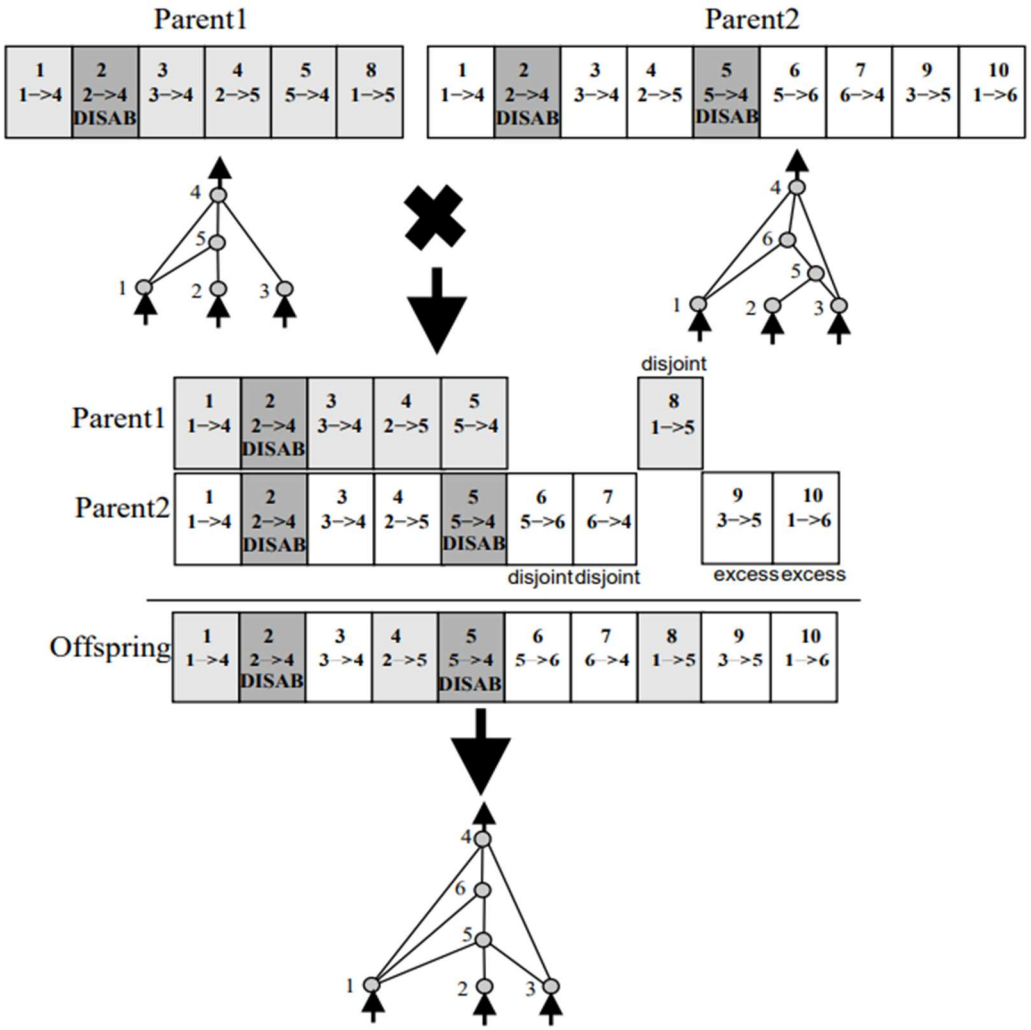
konvencija, s brojnim različitim topologijama i kombinacijama težina, je neizbježni rezultat dopuštanja genomima da rastu bez ograničenja [21]. Sljedeća cjelina objašnjava kako NEAT rješava problem križanja genoma različitih veličina na razuman način.

5.1.1.5. Praćenje gena korištenjem povijesnih oznaka

Koristeći se znanjem o povijesnom podrijetlu svakog gena u populaciji je moguće saznati točno koji geni su međusobno odgovarajući unutar cijele populacije [21]. „Dva gena s istim povijesnim podrijetlom moraju predstavljati istu strukturu, iako je moguće da imaju drukčije težine, zato što se oba deriviraju od zajedničkog gena predaka iz nekog vremena u prošlosti“ [20, p. 656]. Dakle, sve što je nekom sustavu potrebno da zna koje gene poravnati s kojima je pratiti povijesno podrijetlo svakog gena u sustavu.

„Praćenje povijesnih podrijetla ne zahtijeva puno računanja. Kad god se pojavi novi gen (kroz proces mutacije strukture) povećava se globalni broj inovacije i dodjeljuje se tome genu“ [27, p. 36]. Tako brojevi inovacije predstavljaju kronologiju pojava svih gena u sustavu. Kao primjer, može se uzeti slučaj da su se dvije mutacije prikazane na slici broj 12 dogodile jedna za drugom u nekom sustavu. Novom veznom genu koji je stvoren u prvoj mutaciji je dodijeljen broj 7, a dvoje veznih gena koji su dodani tijekom mutacije novog čvora su dodijeljeni brojevi 8 i 9. U budućnosti, „kad god se ti genomi budu križali, njihov potomak će naslijediti iste brojeve inovacije koji su na svakom genu, jer se inovacijski brojevi nikada ne mijenjaju“ [20, p. 656]. Dakle, zahvaljujući povijesnim oznakama sustav može pratiti kada se koji gen pojavio u populaciji genoma. One također omogućavaju izbjegavanje problema konkurentnih konvencija jer se za svaki gen zna koji mu je gen predak, a s tim znanjem se mogu pronaći genomi istih veličina gdje će geni s istim povijesnim podrijetlom uvijek predstavljati istu strukturu, iako mogu imati drukčije vrijednosti težina.

„Mogući problem je da će iste strukturne inovacije dobiti različite brojeve inovacije u istoj generaciji, ako se slučajno ponove više od jedanput“ [20, p. 656]. „Vođenjem popisa inovacija koje su se pojavile u trenutačnoj generaciji, moguće je osigurati da kada se ista struktura pojavi više od jednom kao rezultat nezavisnih mutacija u istoj generaciji, svakoj identičnoj mutaciji se dodjeljuje isti broj inovacije“ [21, p. 108]. Također vrijedi napomenuti kako „se može resetirati listu sa svakom novom generacijom, jer je to dovoljno da spriječi gomilanje brojeva inovacije“ [27, p. 37]. Imajući to na umu, može se reći da se izbjegava eksplozivni porast broja inovacija korištenjem liste brojeva inovacija kako ne bi došlo do zapisa jednog broja inovacije više puta te se ta lista briše sa svakom novom generacijom.



Slika 13: Podudaranje genoma s različitim mrežnim topologijama koristeći se brojevima inovacije *[21]*

Koristeći se povijesnim oznakama, sustav zna točno koji geni se podudaraju s kojima, kao što se može vidjeti na slici broj 13. „Tijekom procesa križanja geni u oba genoma koji imaju iste brojeve inovacije su međusobno poravnani. Takvi geni se zovu podudarni geni“ [21, p. 108]. „Geni koji se ne podudaraju su ili disjunktni ili suvišni, ovisno o tome da li se pojave unutar ili izvan raspona brojeva inovacije drugog roditelja“ [20, p. 656]. „Disjunktni geni su oni koji ne dijele povijesno podrijetlo s nijednim drugim genom u drugom genomu. Dok su suvišni geni oni koji su se pojavili kod jednog roditelja kasnije od bilo kojeg gena u drugom roditelju“ [27, p. 38]. Dakle, ako se gleda primjer na slici broj 13, prvo je potrebno odrediti disjunktne gene. Oni su u tom slučaju geni s inovacijskim brojevima 9 i 10, jer zadnji gen od drugog roditelja ima inovacijski broj 8 koji je manji od 9 i 10, stoga prvi roditelj ne može imati suvišne gene u ovom slučaju. Nakon toga je jednostavno odrediti disjunktne gene, samo je potrebno odrediti koji geni u drugom roditelju imaju inovacijske brojeve koji se ne pojavljuju među genima prvog roditelja, a da nisu već odvojeni u suvišne gene. U ovom slučaju su to geni s inovacijskim

brojevima 6 i 7 za drugog roditelja i gen s inovacijskim brojem 8 za prvog roditelja. „U sastavljanju potomka geni su nasumično odabrani od bilo kojeg roditelja što se tiče podudarnih gena, dok se svi suvišni i disjunktni geni uvijek uzimaju od roditelja koji ima veću spremnost“ [20, p. 656]. „Na ovaj način povijesne oznake dopuštaju NEAT-u da vrši križanje koristeći se linearnim genomima bez potrebe za skupom topologijskom analizom“ [21, pp. 108-109].

Unatoč tome što roditelji na slici broj 13 izgledaju drugačije, brojevi inovacije njihovih gena poručuju koji se geni jednog roditelja podudaraju s kojim genima od drugog roditelja. Koristeći se procesom križanja i inovacijskim brojevima, moguće je kreirati novu strukturu koja kombinira dijelove oba roditelja koji se poklapaju i njihove različite dijelove, a sve to bez korištenja topologijske analize. Podudarni geni se nasljeđuju nasumično, dok se disjunktni geni i suvišni geni nasljeđuju od roditelja koji ima veću razinu prikladnosti. U ovom slučaju pretpostavlja se kako „oba roditelja imaju istu razinu prikladnosti pa se i disjunktni i suvišni geni nasljeđuju nasumično“ [21, p. 109]. Također je potrebno napomenuti kako „postoji šansa da se isključeni geni pojave opet u budućim generacijama i da postoji predodređena šansa da neki naslijeđeni gen bude isključen, ako je isključen u ijednom od roditelja“ [27, p. 38].

„Dodavanjem novih gena u populaciju i smislenim parenjem genoma koji predstavljaju drukčije strukture, sustav može sastaviti populaciju raznolikih topologija“ [21, p. 109]. Nažalost takva populacija ne može održavati topologijske inovacije samostalno, već ju je potrebno zaštiti od preuranjenog nestanka mreža koje sadrže inovativne strukture [22]. Rješenje je korištenje specijacije za zaštitu populacije, kao što je objašnjeno u sljedećoj cjelini.

5.1.1.6. Zaštita inovacije korištenjem specijacije

„Provođenje populacije kroz proces specijacije dopušta organizmima da se natječu prvenstveno unutar vlastitih niša, umjesto da se natječu s ostatkom populacije“ [27, p. 38]. Tako je moguće podijeliti populaciju u vrste, gdje svi članovi neke vrste imaju slične topologije; što bi ujedno pružilo vremena topologijskim inovacijama za optimizaciju svoje strukture kroz natjecanje s drugim mrežama u relativno sigurnom okruženju [20]. Dakle, ideja je da se populacija razvrsta u vrste prema sličnosti topologija mreža. Time bi se topologijskim inovacijama dalo vremena da se optimiziraju natjecanjem s drugim mrežama iz iste vrste prije natjecanja s ostatkom populacije.

„Broj suvišnih i disjunktnih gena između para genoma služi kao prirodno mjerilo za njihovu udaljenost kompatibilnosti. Što su više dva gena disjunktna to manje dijele evolucijske povijesti i zato su manje kompatibilni“ [21, p. 110]. Dakle, „moguće je mjeriti udaljenost kompatibilnosti δ različitih struktura u NEAT-u kao jednostavnu linearnu kombinaciju broja suvišnih E i disjunktnih D gena, uz prosječnu razliku težina podudarnih gena $\overline{W}$, uključujući onesposobljene gene: $\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}$ “ [27, p. 38]. „Koeficijenti c_1, c_2 i c_3 dozvoljavaju

ažuriranje bitnosti tri faktora, dok je faktor N, broj gena unutar većeg genoma, normaliziran za veličinu genoma. N može imati postavljenu vrijednost od 1 ako oba genoma nisu pretjerano velika, tj. ako se sastoje od manje od 20 gena“ [20, p. 657]. Kao testni primjer za mjerenje udaljenosti kompatibilnosti su poslužili genomi iz slike broj 13. Potrebno je napomenuti kako će se sva tri *c* faktora postaviti na vrijednost od 1, isto kao i vrijednost faktora *N* zato što je broj gena u oba genoma ispod 20. Broj disjunktnih gena u genomima je 3, a broj suvišnih gena je 2, dok je prosječna razlika težina podudarnih gena nemoguća za izračunati zbog nedostatka informacija, ali iskorištena je vrijednost 0.1234. Dakle formula za ovaj slučaj bi glasila sljedeće: $\delta = \frac{1*2}{1} + \frac{1*3}{1} + 1 \cdot 0.1234$ te bi vraćala rezultat s vrijednosti od 5.1234.

Mjerilo udaljenosti δ dopušta specijaciju korištenjem ograničenja kompatibilnosti δ_t tako da se svaki genom testira s nasumično odabranim članom određene vrste, i ako testiranje vrati udaljenost između genoma koja je manja od ograničenja kompatibilnosti, onda se genom koji se testirao pridružuje toj vrsti [20]. Održava se uređen popis vrsta u redoslijedu u kojem su se pojavile te se u svakoj generaciji genomi sekvencijalno razvrstavaju u vrste [27]. Zahvaljujući popisu vrsta iz prethodnih generacija, moguće je da „sustav izbaci one vrste koje unatoč prolasku broja generacija, nisu ostvarile pomak u prikladnosti članova vrste“ [20, p. 657]. „Svaka postojeća vrsta je predstavljena s nasumičnim genomom koji je pripadao toj vrsti u prethodnoj generaciji. Neki genom g u trenutačnoj generaciji je svrstan u prvu vrstu u kojoj je g kompatibilan sa genomom koji predstavlja tu vrstu [21, p. 110]“. Na ovaj način se izbjegava preklapanje između vrsta. „Ako g nije kompatibilan sa bilo kojom od postojećih vrsta, onda se stvara nova vrsta kojoj je g predstavnik“ [27, p. 39]. Dakle, i u slučaju gdje je tek početak te ne postoje vrste u koje bi se neki genomi svrstali, prvi genom će biti svrstan u vlastitu vrstu te će se od genoma koji ne pripadaju u tu vrstu opet krenuti stvarati druge nove vrste. Podjelom u vrste zaštićuju se inovacije u strukturama ANN-a i daje im se vremena da sazru i optimiziraju se. Ali, naravno da ako neka vrsta ne napreduje ni nakon nekoliko generacija će ju se izbaciti iz populacije.

Kao mehanizam reprodukcije u NEAT-u koristi se eksplicitno dijeljenje prikladnosti [29], gdje organizmi u istoj vrsti moraju dijeliti prikladnost svoje niše. „Tako vrsta ne može priuštiti postati prevelika, čak i ako mnogi njeni organizmi postižu dobre rezultate. Zato je malo vjerojatno da će bilo koja jedna vrsta preuzeti cijelu populaciju, što je ključno za uspjeh evolucije sa specijacijom“ [20, p. 657]. „Prilagođena prikladnost f_i' za organizam i se računa prema njegovoj udaljenosti δ od svakog drugog organizma j unutar populacije: $f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$ “ [27, p. 39]. Funkcija dijeljenja sh je postavljena na 0 kada je udaljenost $\delta(i,j)$ iznad ograničenja δ_t; inače je $sh\big(\delta(i,j)\big)$ postavljen na vrijednost od 1 [30]. „Dakle, $\sum_{j=1}^{n} sh\big(\delta(i,j)\big)$ se reducira na broj organizama u istoj vrsti kao i organizam i. Ta redukcija je

prirodna jer su vrste već raspoređene u razrede po kompatibilnosti korištenjem ograničenja δ_t. Svakoj vrsti je dodijeljen potencijalno različit broj potomaka koji je proporcionalan sumi ažuriranih spremnosti f_i' njegovih organizma članova“ [21, p. 110]. „Vrste se onda reproduciraju tako što prvo eliminiraju članove s najlošijim rezultatima iz populacije. Nakon toga se cijela populacija zamijeni s potomcima preostalih organizama u svakoj vrsti“ [20, p. 658]. Dakle, reprodukcija se odvija u NEAT algoritmu uz pomoć eksplicitnog dijeljenja prikladnosti, koje znači da svi članovi neke vrste dijele prikladnost te vrste kojoj pripadaju. Zato što kažnjava one vrste koje imaju prevelik broj članova, izbjegava se problem prevladavanja jedne vrste u populaciji. To znači da se štiti strukturna inovacija, jer takve mreže isto dobiju broj članova koji im se pridruži u vrsti te tako spriječi opasnost od njihovog odbacivanja prije nego stignu optimizirati svoje nove strukture.

5.1.1.7. Minimiziranje dimenzionalnosti koristeći se inkrementalnim rastom od minimalne strukture

„Uobičajeno TWEANN-ovi započnu s inicijalnom populacijom od nasumičnih topologija kako bi uveli raznolikost od početka“ [20, p. 658], dok nasuprot tome, NEAT algoritam čini pretraživanje sklonije prostorima minimalnih dimenzija tako da „započne s jednolikom populacijom mreža bez skrivenih čvorova“ [19, p. 2], gdje su svi ulazi spojeni direktno s izlazima, a koje se razlikuju jedino po težinama mreža. „Nova struktura se uvodi postepeno kako se događaju mutacije strukture i preživljavaju samo one strukture koje se pokažu korisnima tijekom procjena spremnosti. Drugim riječima, razrade strukture koje se događaju u NEAT-u su uvijek opravdane“ [21, p. 111]. Zbog toga što populacija započinje minimalno, dimenzionalnost prostora pretraživanja je minimizirana i NEAT uvijek pretražuje manji broj dimenzija od ostalih TWEANN-ova i neuroevolucijskih sustava koji imaju fiksnu topologiju. Minimiziranje dimenzionalnosti daje NEAT-u prednost u izvedbi naspram drugih pristupa [27].

Zahvaljujući činjenici da počinje s minimalnom populacijom, jednolikih ANN-ova, koji su različiti samo po težinama mreža, a isti po tome što nijedan nema skrivenih čvorova, NEAT algoritam može učiti bez brige o razvoju previše kompleksnih mreža za neki zadatak, jer su sve strukture koje su nadodane na osnovu korisne. Također, osiguravanjem prostora za pretragu s minimalnim brojem dimenzija uzrokuje drastične skokove prema naprijed u performansama NEAT-a, naspram nekih drugih algoritama neuroevolucije.

6. Analiza korištenih trkaćih igara

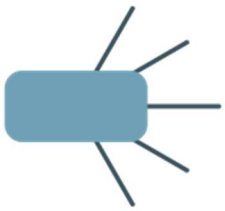
Nakon pruženih definicija i pojašnjenja teorije iza navedenih metoda umjetne inteligencije, rad nastavlja s analizom konkretnih primjera koji su odabrani kao praktični prikazi teorije. Primjeri su dva projekta Watchcarslearn [5] i Micromachine.AI [9] pronađena na stranici GitHub, a koji sadržavaju osnovne elemente trkaće igre i u kojima agent uči voziti po pruženoj trkaćoj stazi koristeći se zadanom metodom strojnog učenja u tom projektu.

U ovom su poglavlju analizirani projekti korišteni u testiranju, a svaki projekt je analiziran u vlastitom odlomku prema pruženom redoslijedu. Ponajprije su navedene komponente svakog projekta, sa kratkim opisom modela kojim agent upravlja te staze na kojoj agent obavlja testiranje. Sljedeće je prikazan kod koji se koristi pri donošenju odluka od strane agenta, gdje je to moguće. Nakon toga, analizira se uspješnost agenta prema parametru brzine prolaza kroz stazu u određenim iteracijama razvoja kod projekta Watchcarslearn, dok se za projekt Micromachine.AI koristi parametar grešaka učinjenih tijekom vožnje. Prije zaključka, iznesene su razlike između korištena dva projekta i razlike u pristupu analizi koje su iz njih proizišle. Rad završava usporedbom između dva tipa strojnog učenja korištena u pojedinim projektima.

6.1. Analiza projekta Watchcarslearn

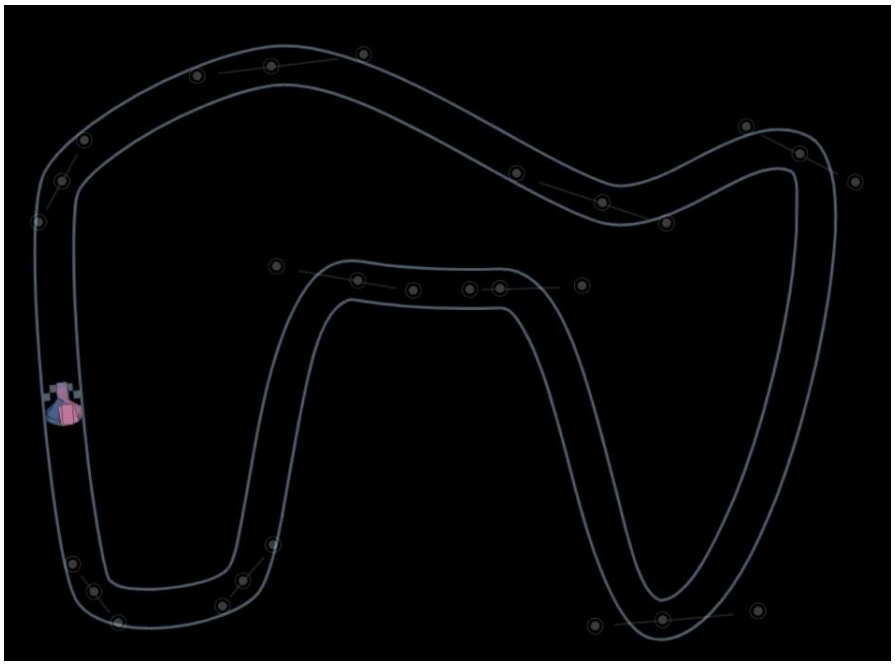
Prvi projekt koji je odabran za analizu je Watchcarslearn od autora Manas Sarpatwar [5], gdje agent uči upravljati autom, koristeći se učenjem NEAT algoritmom, kako bi prošao zadanu stazu i ponovno prešao početnu liniju.

Objekti koji se koriste u projektu sadržavaju broj auta kojima agent upravlja u svakoj generaciji, stazu promjenjivu od strane korisnika, graf koji prati uspješnost auta u završetku kruga po stazi te grafički prikaz rada neuronske mreže tijekom vožnje trenutne generacije agenta.



Slika 14: Prikaz modela auta iz projekta Watchcarslearn *[5]*

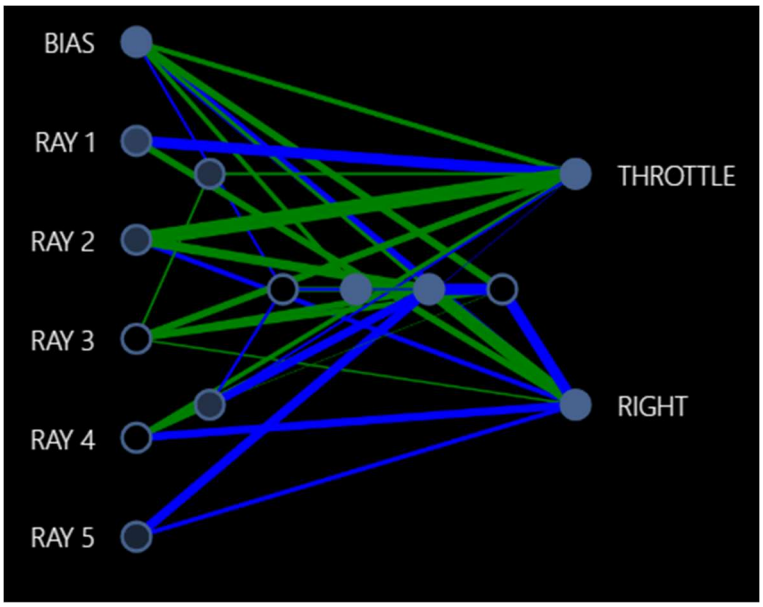
Model auta koji se koristi svake generacije je izgledom samo zaobljeni kvadrat nasumično odabrane boje i agent određuje njegovu poziciju u prostoru uz pomoć 5 senzora za udaljenost koji su raspoređeni na prednjoj strani auta. Oni služe kao podatkovni ulazi za neuronsku mrežu koja kontrolira auto. Svaki od senzora ima unaprijed određenu maksimalnu udaljenost, a podaci koje prosljeđuje se sastoje od normaliziranih vrijednosti ulaza senzora. Te vrijednosti se dobivaju dijeljenjem udaljenosti na kojoj je senzor aktiviran sa njegovom maksimalnom udaljenosti, a neuronska mreža ih koristi kako bi odredila trenutačnu poziciju auta na stazi. Neuronska mreža, koju autor projekta također naziva i „mozgom“ auta, ima dva izlaza, od kojih oboje imaju vrijednosti između 0 i 1, a koji kontroliraju kretanje auta kroz stazu. Prvi izlaz služi za upravljanje kočnicama i gasom auta, gdje u slučaju da je vrijednost dobivena veća od 0.66 auto ubrzava, a u slučaju da je vrijednost manja od 0.33 onda auto usporava. Također ima raspon između tih dviju vrijednosti u kojemu auto održava svoju trenutačnu brzinu. Drugi izlaz nad kojim mozak ima kontrolu određuje smjer prema kojem se prednji dio auta okreće, gdje u slučaju da je vrijednost veća od 0.66 auto skreće prema desno, a u slučaju da je vrijednost manja od 0.33 auto skreće prema lijevo. Kao i s prvim izlazom, za drugi izlaz isto ima raspon između vrijednosti od 0.33 i 0.66 gdje auto ne skreće već drži postojeći smjer do sljedeće promjene.



Slika 15: Prikaz prilagođene staze koja je bila korištena u projektu Watchcarslearn

Staza na kojoj se auti kreću tijekom vožnje se sastoji od velikog broja Bezierovih krivulja od kojih su sastavljene vanjske i unutarnje granice. Beziers krivulje su definirane skupom kontrolnih točki koje određuju oblik koji će krivulja imati. Krivulja se kreira provlačenjem ravne

linije od početne točke do zadnje točke, dok druge kontrolne točke utječu na kako će se krivulja savinuti dok prolazi blizu njih [31]. Također, postoji sakriveni put na sredini staze, koji je nevidljiv korisniku, ali ga neuronska mreža koristi kako bi odredila da li je auto ispravno orijentiran i još uvijek na stazi ili je došlo do sudara. Udaljenost senzora se mjeri na svakom koraku fizike, to jest, mjeri se kada god prođe određeni vremenski interval, a mjeri se računanjem točke sjecišta linije senzora i najbliže Bezierove krivulje. Stazu je moguće mijenjati u potpunosti, čak i tijekom vožnje auta po njoj, sa dodatnom opcijom kreiranja nasumične staze za čije je kreiranje autor koristio Voronoi dijagrame. Dok se staza mjenja, auti koji prolaze stazom su zaustavljeni kako bi se izbjeglo rušenje projekta, a za promjenu staze je potrebno samo kliknuti ikonu olovke postavljenu u gornji desni kut ekrana.



Slika 16: Prikaz korištene neuronske mreže iz projekta Watchcarslearn

U donjem lijevom dijelu ekrana je autor projekta smjestio grafički prikaz rada neuronske mreže i promjena veza u mreži tijekom generacija. Mreža koja je korištena je unaprijedna (engl. feed forward) neuronska mreža koja zaustavlja mutiranje povratnih veza, ona se aktivira koristeći se listom čvorova koji su sortirani topologijski koristeći se Kahnovim algoritmom. Razina sjaja svakog čvora je indikator aktivacije izlaza tog čvora, dok se veze među čvorovima prikazuju zelenom bojom ako imaju pozitivnu težinu, plavom bojom ako imaju negativnu težinu, a crvenom bojom ako je ta veza onemogućena. Isto tako, kada se aktivira izlaz od nekog čvora, onda veza između tog i sljedećeg čvora je podebljana.

Projekt je napisan gotovo isključivo u JavaScript-u, ali su korišteni HTML i CSS za kreiranje vizualne strane projekta. Nije bilo moguće doći do razumljivog koda projekta zato što je projekt prošao proces tijekom kojega je njegov kod stavljen u šifrirani paket(engl. „bundle“)

te ga nije bilo moguće dešifrirati. To je, također, razlog zbog kojega je korištena ručna štoperica za mjerenje vremena potrebnog autu da prođe jedan krug kroz stazu.

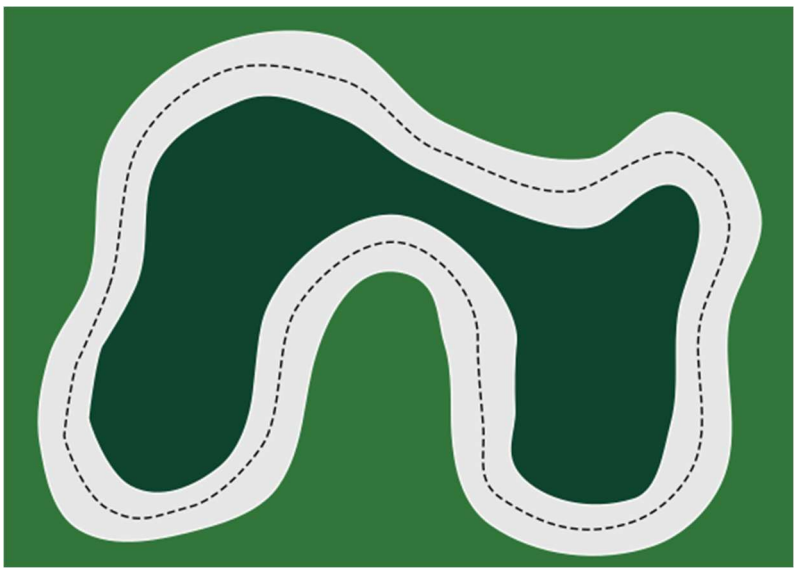
U svakoj generaciji ima više auta na stazi koji pokušavaju doći do cilja, ali auti se ne mogu međusobno sudariti, niti imati ikakve druge međusobne interakcije. Auti se kreću od startne linije prema dolje i u smjeru suprotnome onome kazaljke na satu. Oni auti koji ostvare najbolje rezultate su uključeni u grupu auta koji se voze stazom u sljedećoj generaciji. Kako je rastao broj generacija, tako su agenti općenito krenuli gubiti sve manji broj auta iz svoje grupe, ali im je opala brzina prolaza kroz stazu. Zbog toga što nije moguće pauzirati prolazak auta kroz stazu, prve dvije-tri generacije nisu potpuno pouzdane, jer se staza morala preurediti tako da naliči stazi iz drugog projekta koji je analiziran. Dok se auti zaustave dok staza prolazi promjene, čim se otpusti klik miša nad dijelom staze koji se mijenjao, oni se odmah pokreću. Zbog toga iako je prva generacija ostvarila najbolje vrijeme od 12.30 sekundi, iako većina auta nije prošla kroz stazu, to vrijeme se moralo odbaciti jer se staza mijenjala dok su auti prolazili njome. Sljedeća generacija koja je zabilježena je deseta, gdje je autima bilo potrebno 17.01 sekundi, a u kojoj je naspram prijašnjih generacija manji broj auta skrenuo sa staze ili je nije završio. U dvadesetoj generaciji autima je bilo potrebno 23.20 sekundi kako bi završili krug, ali je napomenuta jer su se auti krenuli kretati u jednoj većoj grupi. Oko četrdesete generacije, većina auta koji ne završe stazu se zabiju na trećem zavoju staze, a samo dva auta uspijevaju završiti stazu. Autima pedesete generacije je bilo potrebno 19.23 sekunde kako bi završili krug, dok auti koji su generirani u pedesetoj generaciji nisu uspjeli doći dalje od prvog zavoja. Od pedeset i četvrte generacije, generacije postaju sve uspješnije u završavanju staze. Osamdeset i peta generacija je bitna po tome što je u njoj bilo vrlo malo sudara, te je svim autima trebalo samo 15.36 sekundi kako bi završili stazu. Kod generacije broj sto je bilo potrebno 20 sekundi kako bi svi auti završili krug, a također je broj auta koji nije završio bio veći. U sto dvadeset i petoj generaciji je većina auta prošla stazu za 17.39 sekunda, dok su većina auta prošla stazu u zadnjem testu, generaciji sto pedeset, ali za tek 20.67 sekundi.

6.2. Analiza projekta Micromachine.AI

Drugi projekt odabran za analizu je „Micromachine.AI“ od autora Cédric Bovar u kojemu agent uči upravljati autom koristeći se umjetnim neuronskim mrežama, ali je potrebno pružiti mu barem određenu minimalnu količinu početnih podataka prema kojima bi započeo učiti procjenjivati kako se ponašati u kojim situacijama.

Objekti koji se koriste u projektu sadržavaju prikaz staze ispred auta koji je vidljiv agentu, a nalazi se u gornjem desnom kutu, stazu koja nije promjenjiva, model auta koji se

koristi zajedno sa njegovim vidnim poljem i indikatorom broja snimljenih situacija prema kojima će agent učiti.



Slika 17: Prikaz staze iz projekta Micromachine.AI *[9]*

Staza je sastavljena od bijele ceste s isprekidanim crnim linijama na njenoj sredini, a okružuju ju zelene površine, tako da agent koji upravlja autom može lako razaznati razliku između ceste i njezine okoline. Također je površina koju staza okružuje tamnije zelene boje kako bi postojala vidljiva razlika između vanjske i unutarnje strane staze, koja je sama po sebi podijeljena po sredini s isprekidanim crnim linijama.

Slika 18: Prikaz modela automobila iz projekta Micromachine.AI *[9]*

Model auta koji se koristi je zapravo ikona koja se pomiče na slici staze iznad koje se nalazi tako da mijenja svoj položaj na x i y osima. Agent koji upravlja automobilom ima mogućnost kretanja naprijed i unazad tijekom čega se miče konzistentnom brzinom, a također može rotirati automobil po njegovoj osi prema lijevoj ili desnoj strani u inkrementima od tri stupnja. Mozak koji agent koristi za kretanje po stazi se uči tako da stvarna osoba postavi auto u samovoljni broj pozicija na stazi te u tim pozicijama pritisne tipku ctrl zajedno sa tipkom za kretanje naprijed ili jednom od tipki za rotiranje auta lijevo ili desno, nakon čega mozak stvara

kopiju slike staze koja se nalazi ispred auta i smjera auta. Nakon što je zabilježena zadovoljavajuća količina takvih instrukcija, klikne se tipka za razmak koja služi za prebacivanje auta na samostalno učenje gdje će se mozak koristiti kamerom usmjerenom ispred njega kako bi uspoređivao situacije u kojima se nađe sa onima za koje ima spremljene instrukcije te se ponašao kako se igrač ponašao u njima. U svakom trenutku je moguće vratiti mozak nazad u stanje učenja i dodati još instrukcija.

Projekt je u potpunosti napisan u programskom jeziku C# te su ispod prikazani neki od važnijih dijelova koda, koji su onda i pojašnjeni.

```
this.Car = new Car(new NNBrain(cameraGrid.TotalPoints));
```

Iznad je kod koji se koristi kod kreiranja novog auta, a koji se izvodi tijekom pokretanja projekta. Osim što se kreira novi automobil, taj kod također kreira mozak umjetne neuronske mreže koristeći se ukupnim brojem točaka vidljivih kameri koja se nalazi ispred automobila. Taj broj se dobiva množenjem visine i širine kamere te ponovnim množenjem dobivenog broja s brojem dva. Također se treba napomenuti da su visina i širina kamere uvijek 10, tako da je broj ukupnih točaka uvijek 200.

```
public void Train()
        {
            var batchSize = this._trainingSet.Count;
            var input = BuilderInstance.Volume.SameAs(new Shape(1, 1,
this.TotalInputPoints, batchSize));
            var output = BuilderInstance.Volume.SameAs(new Shape(1, 1,
3, batchSize));

            for (var i = 0; i < batchSize; i++)
            {
                for (var j = 0; j < this.TotalInputPoints; j++)
                {
                    input.Set(0, 0, j, i,
this._trainingSet[i].Item1[j]);
                }

                output.Set(0, 0, (int)this._trainingSet[i].Item2, i,
1.0f);
            }

            var trainer = new SgdTrainer(this._network) { LearningRate =
0.1f, BatchSize = batchSize };

            float previousLoss;
```

```
        do
        {
            previousLoss = trainer.Loss;
            trainer.Train(input, output);
            Debug.WriteLine(trainer.Loss);
        } while (Math.Abs(previousLoss - trainer.Loss) > 0.01);

        this.Loss = (float)Math.Round(trainer.Loss, 2);
    }
```

Neuronska mreža se trenira tako da prvo stvori ulazne i izlazne volumene te kreira trenera koji se koristi metodom stohastičkog spusta gradijenta, a za kojeg je unaprijed određena brzina učenja i veličina uzorka prema kojemu uči. Nakon toga, trener će trenirati mrežu skroz dok ne dođe do konvergencije gubitka između prijašnjeg gubitka i trenutnog gubitka trenera.

Zbog konstantne brzine auta, brzina prolaza auta kroz stazu, to jest vrijeme potrebno autu da napravi puni krug od prijelaza preko početne linije, nije bilo moguće koristiti kao glavni parametar za uspoređivanje uspješnosti generacija. Zato je fokus premješten na usporedbu uspješnosti učenja agenta ovisno o količini instrukcija koje su mu dostupne.

Testiranje je provedeno na slučajevima kada je pružena samo jedna uputa, kada je pruženo 5 uputa, kada je pruženo 10 uputa, kada je pruženo 15 uputa i slučaj kada nije pružena niti jedna uputa. Upute koje se spominju su zapravo početni skup parova na kojima agent uči, a koje mu pruža tester prije nego što agent počinje učiti. Tester zadaje te funkcije koristeći se tipkama ctrl i jedne od tipke pokreta kada se nalazi na željenoj poziciji i okrenut je u željenom smjeru. Dakle, 1 uputa bi mogla izgledati kao što je prikazano ovdje: (slika pozicije u kojoj se auto nalazio; zadani smjer kretanja je lijevo). Također, vrijedi napomenuti kako je auto smješten u istu poziciju i smjer kao i na prethodnom projektu, te da je u prvom slučaju također testirano kako se auto ponaša ovisno o kojoj je danoj uputi riječ, bila ona slučaj kada se auto pomiče ravno, lijevo ili desno. Bez uputa iz kojih bi učio, agent nasumično kruži stazom, prelazi njene rubove te općenito ne može naučiti išta. U slučaju da mu je jedna uputa kada se kretati ravno onda je agent sposoban samo pomaknuti se prema naprijed od svoje početne pozicije te ubrzo završi izvan područja staze. Također, u slučajevima kada je agentu samo pružena uputa za kada skrenuti lijevo ili desno, agent kreće raditi krugove oko svoje osi i ovisno o tome je li uputa za desno ili lijevo, kreće se okretati u tome smjeru. Kada su mu pružene 5 instrukcija, dvije od kojih su za slučajeve kada se treba kretati ravno, dvije za kada treba skrenuti lijevo i jedna za kada treba skrenuti desno, agent uspijeva napraviti krug, ali pritom često izlazi iz naznačene ceste na svijetliju zelenu površinu te pomalo nasumično skreće lijevo

i desno na stazi kako bi se izravnao. Već sa deset instrukcija agent se pomiče po stazi puno lakše, ne silazi sa jasno označenog puta, iako u nekim slučajevima dolazi blizu unutarnjeg ruba te nije potpuno izgubio naviku povremenog iznenadnog skretanja lijevo i desno na ravnim dijelovima ceste. S 15 instrukcija agent se ponaša iznimno dobro na stazi, ne prelazi preko rubova ceste i rješio se problema iznenadnog skretanja tijekom ravnih dionica ceste.

Također su testirani slučajevi u kojima je korišten agent s početnim skupom od pet instrukcija, gdje su promijenjeni dijelovi koda. U prvome slučaju je promijenjena vrijednost koja određuje koliko vremena agent provede trenirajući prije nego ga sustav smatra optimiziranim. Vrijednost koja se mijenja je apsolutna razlika između vrijednosti trenutne funkcije pogreške i vrijednosti prošle funkcije pogreške, gdje se agent trenira skroz dok ta razlika ne bude manja od 0.01, kao što je originalno određeno. Koristit će se skraćenica vrijednost PD kako bi bilo preglednije čitati rad. Testirana su ponašanja agenta u slučaju da je vrijednost PD smanjena na 0.005 i 0.001, a u slučajevima da je vrijednost PD povećana su se koristile vrijednosti 0.015 i 0.05. Korišten je referentni model koji ima isti skup instrukcija, ali s vrijednosti PD od 0.01. Opis njegovog ponašanja na stazi je moguće pronaći u paragrafu iznad.

Prvo je testiran slučaj u kojemu je vrijednost PD smanjena s originalne vrijednosti 0.01 na 0.005. Nakon što su agentu pružene početne instrukcije, agent se krenuo pomicati po stazi kako bi završio krug, ali kada je došao do izašao iz prvog velikog skretanja desno, presjekao je stazu vozeći se po tamno zelenoj unutrašnjosti kruga. To je ponašanje ponovio kada je došao do početka zavoja prema lijevo koji se nalazi na gornjoj desnoj strani staze, gdje je naglo skrenuo lijevo te vozio se ravno po tamno zelenoj unutrašnjosti kruga dok nije opet izašao na početak staze. Agent je napravio tri kruga u kojima nije poboljšao svoje ponašanje. Drugi slučaj koji je bio testiran je imao drugo smanjivanje vrijednosti, sa 0.01 na 0.001. Unatoč lošim rezultatima prethodnog testiranja, agent je u ovom testu vozio bolje nego referentni model. Nije bilo zapinjanja za rubove staze ili prijelaza kroz unutrašnjost kruga.

Sljedeće su testirani slučajevi u kojima je vrijednost PD povećana na 0.015 i 0.05. Kod testiranja slučaja s vrijednosti PD od 0.015 nisu uočene velika odstupanja od ponašanja agenta referentnog modela. Agent osim što zna blago prelaziti na svjetliju zelenu površinu vanjske strane staze također zapinje za rubove unutrašnje tamno zelene dijelove kruga. U slučaju s vrijednosti PD od 0.05 agent je prilikom ulaza u prvi zavoj se primaknuo unutrašnjosti staze te ju krenuo blago gaziti. Tijekom ostatka njegove vožnje, agent se nije odvajao od unutrašnjosti staze, osim u slučaju velikog obrnutog U-zavoja koji se nalazi na sredini staze. Tada se odvojio od ruba staze te vozio normalno, ali se odmah po završetku tog zavoja vratio nazad u poziciju tik uz unutrašnjost.

6.3. Diskusija analiziranih projekata

Prije donošenja zaključka o prednostima korištenih algoritama strojnog učenja u ovim projektima, opisane su njihove razlike i kako su one utjecale na analiziranje tog specifičnog projekta. Također je time prikazano zašto nije bilo moguće koristiti se zajedničkim parametrom prema kojemu bi se mogli uspoređivati korišteni algoritmi.

U projektu Watchcarslearn od autora Manas Sarpatwar [5], velike razlike naspram drugog projekta su bile to što se u jednoj generaciji koristi više od jednog auta, što se vraćaju konkretni podaci i to što su prisutni grafovi koji se osvježavaju sa svakom generacijom, a zbog kojih je bilo puno jednostavnije napraviti analizu učenja naspram drugog projekta. Unatoč svojih prednosti projekt Watchcarslearn je bio znatno kompliciraniji za pripremiti za testiranje. Poteškoće su proizašle iz toga što je bilo potrebno preurediti stazu koja se koristila tako da ona bude što sličnija stazi zadanoj u projektu Micromachine.AI, što je postalo problematično kada se ispostavilo da nije moguće pauzirati aute, to jest zaustaviti testiranje bez gubitka promjena učinjenih nad stazom. Zbog toga, i nemogućnosti resetiranja učenja automobila nazad na prvu generaciju bez promjene staze, se moralo odbaciti prvih nekoliko generacija dok se staza nije preuredila u željeni oblik. Također nije bilo lako pratiti aute iz samo jedne generacije zbog preklapanja tih auta sa onima iz drugih generacija što je znalo dovesti do manjih fluktuacija u zapisu vremena potrebnog za prolazak kroz stazu. Još jedan problem je bio nemogućnost pristupa kodu korištenom u projektu, zato što ga je autor proveo kroz proces koji ga je pretvorio u „bundle“ oblik koda, što je rezultiralo šifriranim kodom kojega nije bilo moguće dešifrirati kako bi se moglo pobliže analizirati kod NEAT algoritma korištenog u projektu.

Projekt Micromachine.AI od autora Cédric Bovar [9] je bio puno jednostavniji za pripremiti za testiranje, jer je bilo potrebno samo pokretanje projekta u razvojnom okruženju Visual Studio, vožnja auta na početnu poziciju i zadavanje određenog broja instrukcija po testu. Također je bitno napomenuti kako je kod bio jasan, pregledan i sadržavao komentare od autora koji su pojašnjavali dijelove koda. Zbog tog razloga u analizi je bilo moguće nadoknaditi inače oskudan opis projekta s informacijama dobivenim razumijevanjem koda. Do problema je došlo kada je uspostavljeno da auto koji se pomiče po stazi čini to konzistentnom brzinom. To je uzrokovalo manjak konkretnih rezultata analize i zahtijevalo da se parametar usporedbe promijeni na uspješnost prolaza stazom bez prijelaza preko označenih rubova staze. U ovom projektu također ne postoje jasno naznačene generacije, pa je testiranje podijeljeno prema tome koliko je instrukcija zadano automobilu na početku testiranja.

6.4. Zaključak o analiziranim metodama i algoritmima

Tijekom rada s projektima i proučavanja teorije iza metode učenja uz pomoć neuronskih mreža te NEAT algoritma, stekao sam dojam kako postoje sličnosti između ta dva načina učenja u tome što se NEAT algoritam koristi evolucijom populacije genoma, svaki od kojih predstavlja neku neuronsku mrežu. Također su slični po tome što se koriste funkcijom pogreške kako bi odredili uspješnost rješavanja traženog zadatka. Unatoč tim sličnostima postoji i nekoliko ključnih razlika koje sam primijetio tijekom rada s projektima u kojima su implementirani. Prva očita razlika između njih je razlika u topologiji. Dok neuronske mreže uobičajeno imaju fiksne topologije, koje je potrebno odrediti unaprijed, NEAT algoritam razvija topologije svojih mreža tijekom procesa treniranja, dodavanjem novih čvorova i veza te odbacivanjem onih koje nisu korisne. Zato je u projektu MicroMachine.AI bilo potrebno unesti početne podatke na kojima će se agent trenirati prije svakog testiranja. Dok u Watchcarslearn projektu to nije bilo potrebno, ali je imao problem što projekt nije imao opciju za pauziranje procesa testiranja. Tijekom testiranja sam primijetio da u slučajevima kada im je od početka pružena velika količina podataka neuronske mreže znaju biti znano brže u dostizanja zadovoljavajuće razine uspješnosti od NEAT algoritma. Nažalost, isto se ne može reći u slučajevima kada im je pružen manji broj podataka. Tada je dolazilo do slučajeva u kojima je agent mogao samo se pomicati u jednom smjeru ili se uopće nije držao definirane staze. Nasuprot toga, NEAT algoritam je pokazao da može biti vrlo uspješan bez ikakvih početnih podataka, te su relativno brzo neki od auta iz populacije radili krugove u impresivnom vremenu. Pošto ih nije bilo moguće uspoređivati prema brzini zbog konstantne brzine korištene od strane auta u projektu MicroMachine.AI, uspoređivao sam algoritam i metodu po broju grešaka koje su činili tijekom prolaza kroz jedan krug. Unatoč tome što veliki dijelovi populacije nisu preživjeli svaku generaciju u testiranju NEAT algoritma, kako se povećavao broj generacija tako je broj preživjelih rastao jer su radili manje pogreški, dok nije bilo velikog pada u brzini prolaza kroz stazu. Dok u slučaju neuronskih mreži na projektu Watchcarslearn nije bilo velikih promjena tijekom prolaza kroz stazu. Smatram kako unatoč sporijem početku, NEAT algoritam je bolja opcija, jer ne zahtijeva skup početnih podataka, bilo ga je jednostavnije interpretirati i temeljen je na optimiziranju rješenja zadatka, što savršeno odgovara domeni trkaćih računalnih igara, gdje je zadatak uobičajeno što brže proći kroz stazu.

7. Zaključak

U ovom radu su pružene osnove strojnog učenja i analizirane odabrane metode i algoritmi koji su primijenjeni na odabranim projektima Watchcarslearn, autora Manas Sarpatwar [7] i MicroMachine.AI, autora Cédric Bovar.

Kao početak razrade teme je iznesena teorija iza strojnog učenja te metoda i algoritama korištenih u projektima, kako bi se pružila potrebna predznanja za razumijevanje ostatka rada. To poglavlje sadržava definiciju strojnog učenja i problema kojima se ono bavi. Također sadržava podjelu strojnog učenja na dvije osnovne kategorije s kratkim opisom pojedinih kategorija.

Slijede ga poglavlja koja razrađuju odabrane metode i algoritme strojnog učenja, specifično nadziranog učenja i učenja hibridnim algoritmom. Poglavlje o nadziranom učenju započinje s definicijom nadziranog učenja, koja se nastavlja u definiciju algoritma i metoda nadziranog učenja, specifično umjetnih neuronskih mreža, metode koja se koristi u projektu MicroMachine.AI. Nakon pružene definicije neuronskih mreža, pojašnjava se uloga aktivacijskih funkcija u njima, s pobliže objašnjenom najčešćom vrstom aktivacijske funkcije te objašnjenjem algoritma stohastičkog gradijenta, koji se koristi u projektu MicroMachine.AI. Slijede ga objašnjavanja strukture neuronskih mreža te kako točno one uče.

Poglavlje nakon toga počinje definicijom što to znači učenje s hibridnim algoritmima, a nastavlja se u definiciju jednog od najpoznatijih hibridnih metoda neuroevolucije. Osim definiranja neuroevolucije, također je pružen i pogled u kako se odvija proces učenja u toj metodi. Slijedi definicija najpoznatijeg algoritma neuroevolucije, NEAT algoritma. Prvo su opisani problemi neuroevolucije koje je NEAT riješio, te onda pojašnjena rješenja specifičnih problema.

Nakon poglavlja koja su se bavila teoretskim dijelom ovoga rada, su predstavljeni projekti koji su odabrani za analizu. Prvo su opisane njihove komponente, prikazani modeli koji su se koristili u projektu te pokazani dijelovi koda iz projekta, gdje je to bilo moguće. Nakon toga je opisan proces testiranja pojedinačnog projekta te pružena diskusija o radu s njima.

Zaključeno je od strane autora kako je unatoč nekim sličnostima, NEAT algoritam superiorniji za primjenu u domeni računalnih trkaćih igara zbog toga što se lakše snalazi u novim okolinama, nije mu potreban početni skup podataka i temeljen je na optimiziranju rješenja zadatka, što se poklapa s uobičajenim ciljevima trkaćih računalnih igara.

Popis literature

Reference

[1] J. Alzubi, A. Nayyar i A. Kumar, »Machine Learning from Theory to Algorithms: An Overview,« u *Journal of Physics: Conference Series*, Bangalore, India, 2018.

[2] T. Mitchell, Machine Learning, New York: McGraw-Hill Education, 1997.

[3] N. Bolf, »Osvježimo znanje,« *Kemija u industriji,* pp. 591-593, 9 10 2021.

[4] I. Goodfellow, Y. Bengio i A. Courville, Deep learning, The MIT Press, 2016, p. 305–307.

[5] M. Sarpatwar, »Watchcarslearn,« 2021. [Mrežno]. Available: https://awesomeopensource.com/project/manassarpatwar/WatchCarsLearn.

[6] S. Russell i P. Norvig, Artificial Intelligence: A Modern Approach, Global Edition, Pearson, 2021.

[7] J. Togelius i G. N. Yannakakis, Artificial Intelligence and Games, Springer, 2018.

[8] D. Wolpert i W. Macready, »No free lunch theorems for optimization,« *IEEE Transactions on Evolutionary Computation vol. 1, no. 1,* pp. 67-82, Travanj 1997.

[9] C. Bovar, »Micromachine.AI,« 2019. [Mrežno]. Available: https://github.com/cbovar/Micromachine.AI.

[10] J. Togelius, Playing smart : on games, intelligence and Artificial Intelligence, Cambridge, Massachusetts: MIT Press, 2018.

[11] P. Roberts, Artificial Intelligence in Games, Boca Raton: CRC Press, 2023.

[12] M. Saeed, »A Gentle Introduction To Sigmoid Function,« 25 Kolovoz 2021. [Mrežno]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/.

[13] A. V. Srinivasan, »Stochastic Gradient Descent — Clearly Explained !!,« 7 9 2019. [Mrežno]. Available: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

[14] D. Trehan, »Gradient Descent Explained,« 22 5 2020. [Mrežno]. Available: https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c.

[15] R. Bhat, »Adaptive Learning Rate: AdaGrad and RMSprop,« 10 10 2020. [Mrežno]. Available: https://towardsdatascience.com/adaptive-learning-rate-adagrad-and-rmsprop-46a7d547d244.

[16] J. Brownlee, »Gradient Descent With Momentum from Scratch,« 12 10 2021. [Mrežno]. Available: https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/.

[17] Y. LeCun, Y. Bengio i G. Hinton, »Deep learning,« *Nature,* pp. 436-444, 28 Svibanj 2015.

[18] J. McCaffrey, »Visual Studio Magazine,« 18 8 2014. [Mrežno]. Available: https://visualstudiomagazine.com/articles/2014/08/01/batch-training.aspx.

[19] S. Rodzin, O. Rodzina i L. Rodzina, »Neuroevolution: Problems, algorithms, and experiments,« u *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, Baku, Azerbaijan, 2016.

[20] K. O. Stanley, B. D. Bryant i R. Miikkulainen, »Real-time neuroevolution in the NERO video game,« u *IEEE Transactions on Evolutionary Computation, vol. 9, no. 6*, 2005.

[21] K. O. Stanley i R. Miikkulainen, »Evolving Neural Networks through Augmenting Topologies,« *Evolutionary Computation,* pp. 99-127, 2002.

[22] K. O. Stanley, J. Clune, J. Lehman i R. Miikkulainen, »Designing neural networks through neuroevolution,« *Nature Machine Intelligence,* pp. 24-35, 7 Siječanj 2019.

[23] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot i D. Wierstra, »Convolution by Evolution: Differentiable Pattern Producing Networks,« 8 Lipanj 2016. [Mrežno]. Available: https://doi.org/10.48550/arXiv.1606.02580.

[24] P. Pauls, »A Primer on the Fundamental Concepts of Neuroevolution,« 19 Siječanj 2020. [Mrežno]. Available: https://towardsdatascience.com/a-primer-on-the-fundamental-concepts-of-neuroevolution-9068f532f7f7.

[25] S. Risi i J. Togelius, »Neuroevolution in Games: State of the Art and Open Challenges,« u *IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 1* , 2017.

[26] D. J. Montana i L. Davis, »Training Feedforward Neural Networks Using Genetic Algorithms,« u *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, San Francisco, California: Morgan Kaufmann, 1989, pp. 762-767.

[27] K. O. Stanley, »Efficient Evolution of Neural Networks through Complexification,« The University of Texas at Austin, 2004.

[28] S. Bera, »Crossover Operator — The Heart of Genetic Algorithm,« 15 Svibanj 2020. [Mrežno]. Available: https://medium.com/@samiran.bera/crossover-operator-the-heart-of-genetic-algorithm-6c0fdcb405c0.

[29] D. E. Goldberg i J. T. Richardson, »Genetic Algorithms with Sharing for Multimodalfunction Optimization,« u *Proceedings of the Second International Conference on Genetic Algorithms*, San Francisco, California: Morgan Kaufmann, 1987, pp. 148-154.

[30] W. Spears, »Speciation Using Tag Bits,« u *Handbook of Evolutionary Computation*, Oxford, IOP Publishing Ltd. and Oxford University Press, 1995.

[31] E. Alpaydin, Strojno učenje, Zagreb: MATE d.o.o., 2021.

[32] P. Cunningham, M. Cord i S. J. Delany, »Supervised Learning,« u *Machine Learning Techniques for Multimedia*, Heidelberg, Springer Berlin, 2008, pp. 21-49.

[33] I. Goodfellow, Y. Bengio i A. Courville, »Deep learning,« *Genet Program Evolvable Mach 19,* p. 305–307, 29 Listopad 2017.

Popis slika

Popis slika koje se mogu naći u ovome radu.

Slika 1: Podjela metoda strojnog učenja [3] 5
Slika 2:Proširenі prikaz neurona sa aktivacijskom funkcijom[11] 12
Slika 3: Prikaz logističke sigmoidne funkcije 13
Slika 4: Prikaz kako algoritam gradijenta spusta koristi derivacije funkcije za praćenje funkcije do minimuma [4] 14
Slika 5: Prikaz razlike u stopama učenja [14] 15
Slika 6: Prikaz problema zapinjanja u lokalnim minimumima [4] 16
Slika 7: Primjer MLP-a sa tri ulaza, jednim skrivenim slojem koji sadržava četvero skrivenih neurona i dva izlaza [7] 17
Slika 8: Primjer procesa algoritma povratnog rasprostiranja pogreški (engl. backpropagation) [11] 21
Slika 9: Primjer tipičnog generacijskog procesa neuroevolucije[24] 26
Slika 10: Primjer problema konkurentnih konvencija [27] 28
Slika 11: Primjer mapiranja genotipa u fenotip [20] 32
Slika 12: Dva tipa mutacije strukture u NEAT algoritmu [21] 33
Slika 13: Podudaranje genoma s različitim mrežnim topologijama koristeći se brojevima inovacije [21] 35
Slika 14: Prikaz modela auta iz projekta Watchcarslearn [5] 39
Slika 15: Prikaz prilagođene staze koja je bila korištena u projektu Watchcarslearn 40
Slika 16: Prikaz korištene neuronske mreže iz projekta Watchcarslearn 41
Slika 17: Prikaz staze iz projekta Micromachine.AI [9] 43
Slika 18: Prikaz modela automobila iz projekta Micromachine.AI [9] 43